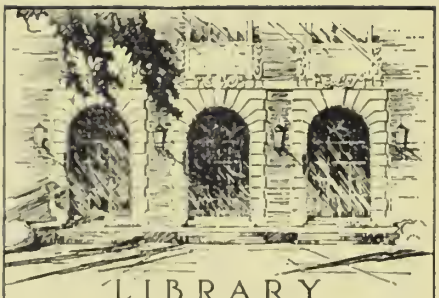


Report No. 301

AN ALGORITHM FOR DELAY CHECKING COMPUTER DESIGNS

by

Jay Merrill Tummelson

January 13, 1969

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 301

AN ALGORITHM FOR DELAY CHECKING COMPUTER DESIGNS*

by

Jay Merrill Tummelson

January 13, 1969

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

* This work was supported in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center under Contract No. US AF 30(602)4144 and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, February, 1969.

ACKNOWLEDGEMENT

The author wishes to express sincere appreciation and gratitude to all those persons whose assistance made this manuscript possible. To Professor D. L. Slotnick, Thesis Advisor, who first aroused my interest in the field of Design Automation and then suggested delay checking as a possible thesis topic. To Mr. Arthur B. Carroll, whose assistance in proofreading and rewording suggestions proved invaluable to the final content of this paper. To Mr. D. O. Pearson, who was an invaluable source of knowledge in Design Automation, specifically in the area of delay checking. To Mrs. Frieda Anderson and Mrs. Mildred Pape, who devoted themselves to typing this manuscript as though it were their own. To Mrs. Shirley Brown, Mrs. Diana Higgs and Mrs. Sharon Hardman who typed the lettering for the flowcharts and who would not settle for less than perfection. Finally, to Mrs. Nancy Stone, Mrs. Dianna Smith, and Mr. Jim Stevens, who painstakingly drew the boxes, diamonds, circles, lines, and arrow on the flowcharts.

ABSTRACT

The problem of delay checking computer designs is discussed along with its relation to the design automation problem as a whole. The ILLIAC IV design automation package is described as an example of systems in general. The remainder of the paper describes in detail the delay check algorithm and computer program developed by the author.

Detailed description of the data formats, internal structuring of data and flowcharts of the program are included for those interested in the application of the algorithm.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 The Design Automation Problem

The function of design automation is to provide designers of electronic equipment assistance in the process of designing and manufacturing this equipment. Since much of the work associated with designing and manufacturing equipment is long, tedious busy work and because people in the computing industry are constantly looking for ways to reduce cost and improve schedules, the use of computers to aid in the design of equipment came quite naturally to the industry. At the optimum level the design automation system permits the designer to input the logic equations which define the machine he wants and the system will generate finished artwork masters, drill tapes, and prints necessary to manufacture the equipment. Unfortunately, not all design automation systems are capable of doing the entire job. Exactly how much of the job can be done by a system is a function of the equipment being designed, the parts available to implement the equipment, the allowable tolerance in the design and, of course, the design automation system itself.

1.2 The ILLIAC IV Design Automation System

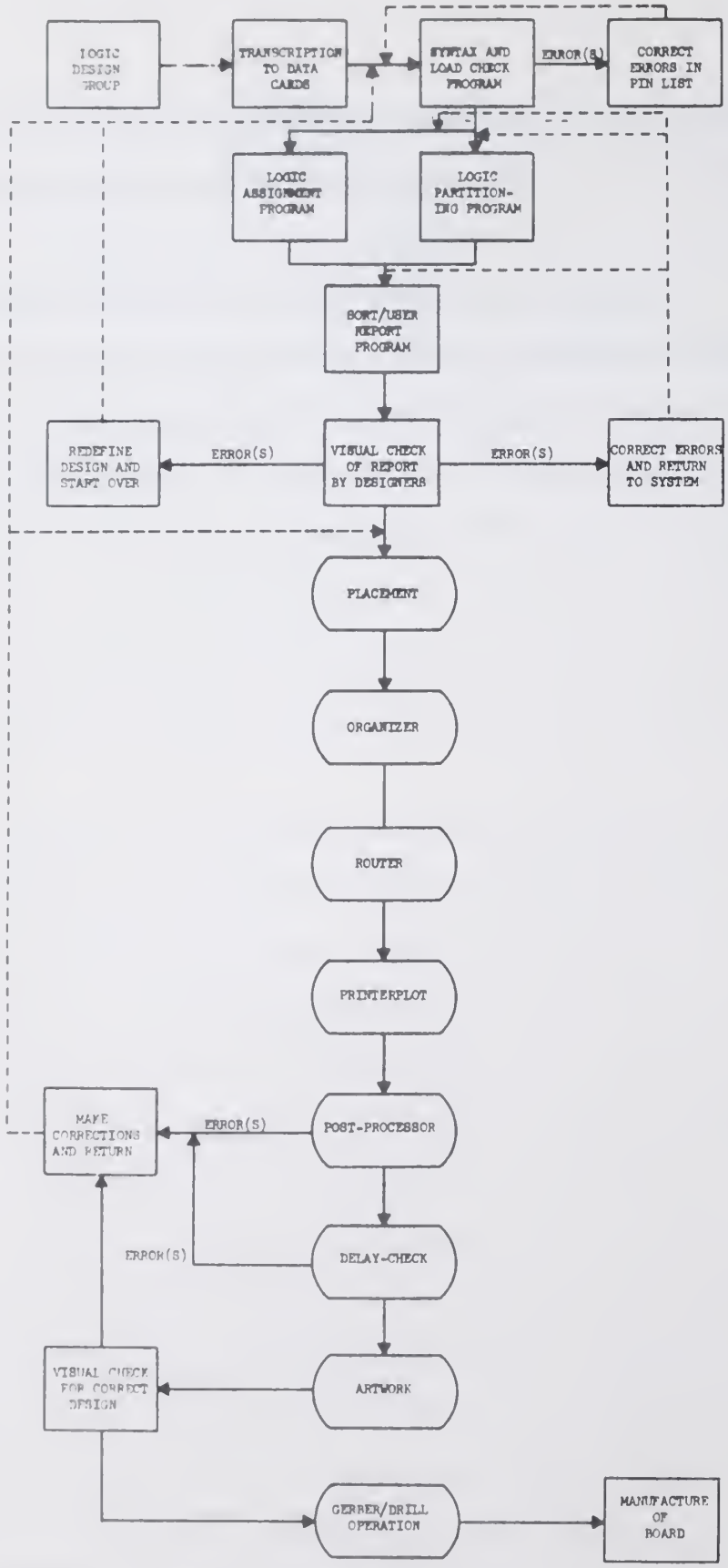
The ILLIAC IV design automation system is one of those which is not capable of doing the entire job. One of the major difficulties is that ILLIAC IV is implemented with the new emitter coupled logic (ECL) for which design specifications and wiring rules are

not completely defined. Naturally, since the designers can not define rigid rules for the use of ECL, a program can not be written to do this. The system must then provide for human interface to correct error situations not anticipated by the designer. This problem has occurred many times before (for example, when transistors first appeared or when printed circuits were first used), and as soon as designers were able to set up rigid rules then design automation was able to catch up with hardware state of the art. The problem is that we are using techniques developed for pre-ECL components to develop an ECL design automation system. It will not work, and we know it, so we must allow for error detection and correction.

Because of the problems described above there are two types of programs in the ILLIAC IV design automation system. There are programs which do the actual computations necessary to design the equipment and there are programs which check the output from the computational programs at various stages to see that all the rules set up to insure correct usage of ECL are being followed. If errors are found, then the checking program will list them so that the designer may correct them before allowing the system to finish this design. This human interplay is an unfortunate but necessary part of this system, and accordingly the system was designed around this aspect. It is, therefore, possible for many part of the system to accept input from previous stages of the system as well as input generated by the designers to update or correct the design. This

means that after the designer corrects an error he can use the system to insure that he has done so correctly without introducing other errors.

The ILLIAC IV design automation system is a set of interconnected computer programs together with provisions for human evaluation and possible updating of intermediate results. A flow-chart for the entire system follows and is followed by a short description of what is done at each step.



ILLIAC IV Design Automation System

LOGIC DESIGN GROUP: The designers make the decision on the specification of the piece of equipment and generate the logic equations which define the machine.

TRANSCRIPTION TO DATA CARDS: The design equations are converted to the format required by the program and are punched in data cards to be input to the program.

SYNTAX AND LOAD CHECK PROGRAM: This program checks the key punched pin list data for proper format. The program also performs a load check for every source pin in the pin list. All errors are noted so that they may be corrected.

CORRECT ERRORS IN PIN LIST: Errors noted above are corrected and the cards are once more input to the syntax and load check program to insure all errors have been corrected.

LOGIC PARTITIONING PROGRAM: This program partitions groups of logic into boards. It is capable of working at the functional level as well as at the physical package level.

LOGIC ASSIGNMENT PROGRAM: This program makes assignment of logic to physical I.C. packages. At the same time it modifies all fields in the records affected by the above logic assignment.

SORT/USER REPORT PROGRAM: Sorts the records of the pin list file as needed by following programs. The second part of this program generates user reports to be checked by designers for error detection and correction.

VISUAL CHECK OF REPORT BY DESIGNERS: At this point, the designers scan the report and check to see that the partitioning and assignment has been done correctly. If not, corrections are made and the design process restarted at the assignment, partition, or sort/user report programs. In the case of severe errors, they may wish to redefine their design and start all over.

PLACEMENT: This program places the components on the board and also arranges the boards on the backplane.

ORGANIZER: This program determines which pins are to be wired together and the sequence in which they are to be wired.

ROUTER: This program routes wires between the pins as specified by the organizer. The output is a record for each segment of the routed wire.

PRINTER PLOT: This program produces a rough sketch on a line printer of the router solution.

POST-PROCESSOR: This program checks the routed wire solution for adherence to designer defined wiring rules. It also checks the loading on each net for correctness and calculates wire delay for each net. Errors flaged by the post-processor should be corrected by the designer. He then decides how far back to go (if the errors are severe enough, he may have to alter the original design equations and start over).

DELAY-CHECK: This program calculates time delay between latches to be checked by the designer for correctness. This is the part of the system that this paper deals with in detail.

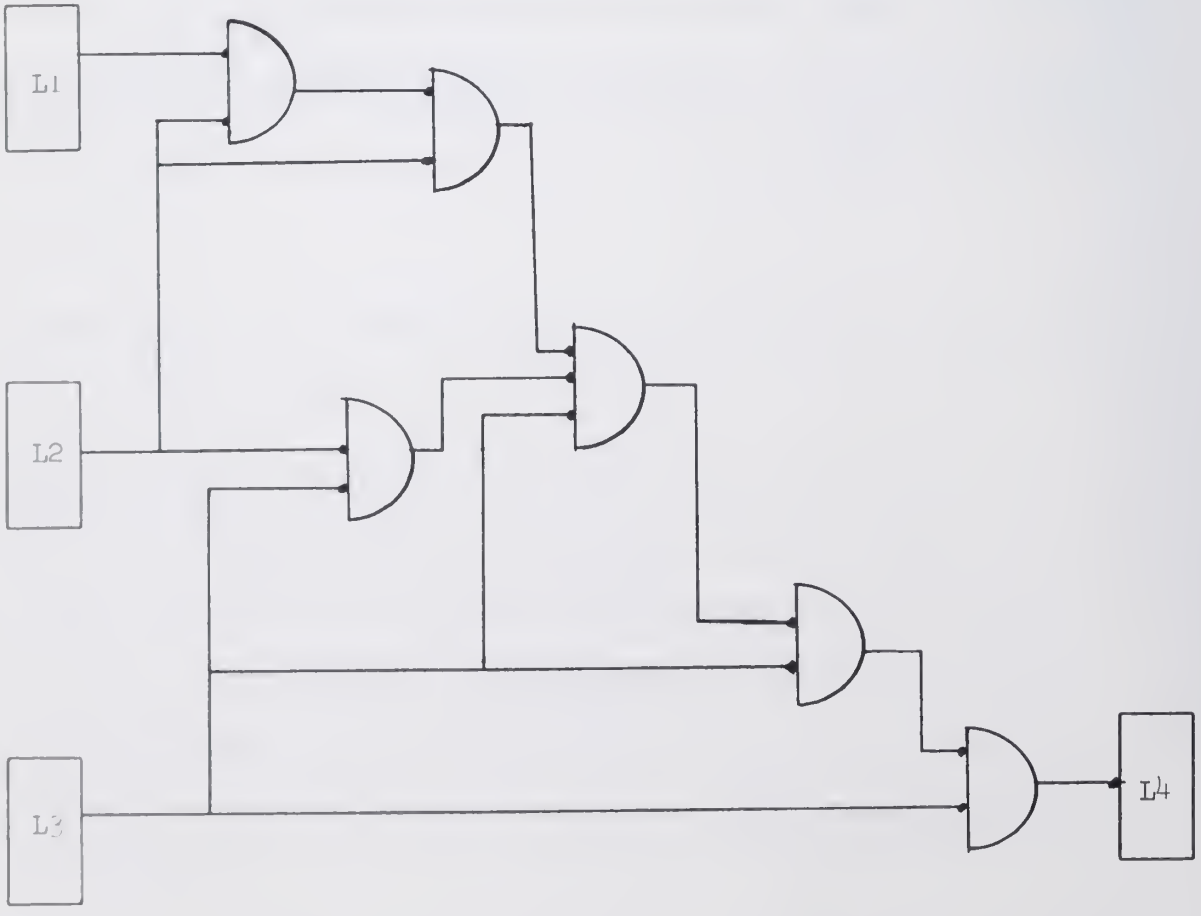
ARTWORK: This program checks the router solution for physical errors, produces tapes for artwork generation on a Gerber Plotter, and produces tapes for driving a numerically controlled drill for drilling the required holes on the production boards.

GERBER/DRILL OPERATION: This section produces the artwork on a Gerber plotter and drills holes on the boards.

MANUFACTURE OF BOARD: This section actually finishes the board. This includes placing of components, printing of wires, and final testing of the board.

2. THE DELAY CHECKING PROBLEM

In a synchronous machine the signals are retimed at a fixed clock rate. This is accomplished by gating the signals into memory elements (latches) after the signal has gone through several combinatorial gates. The amount of combinatorial logic allowed between latches depends upon the speed of the logic and the length of the clock interval. Below is shown an example net which begins at latches L1, L2, and L3 and ends at latch L4.



In the example above when the clock "turns on" the latches L1, L2, and L3, the signals "stored" there are propagated through the combinatorial logic until the "result" finally reaches L4 where it will be stored and "wait" until the next clock pulse comes along to send it from L4 through the next set of combinatorial logic.

This is where delay checking comes in. Designers must know how long it takes for the signal to get from one set of latches to the next. In the example above, the signal from L1 must go through five gates before it reaches L4; however, the signal from L3 may go through only one (though it can also go through two, three and four gates to get to L4). The example above can be used to show the two kinds of errors that designers try to avoid. Suppose the algorithm for calculating the delay along a net was

$$D = W + C^2$$

where W is the number of wires and C is the number of components. Then we can find the following delays:

Longest delay from L1 to L4: 6 wires + 5 components

$$D_1 = 6 + 25 = 31$$

Shortest delay from L2 to L4: 5 wires + 4 components

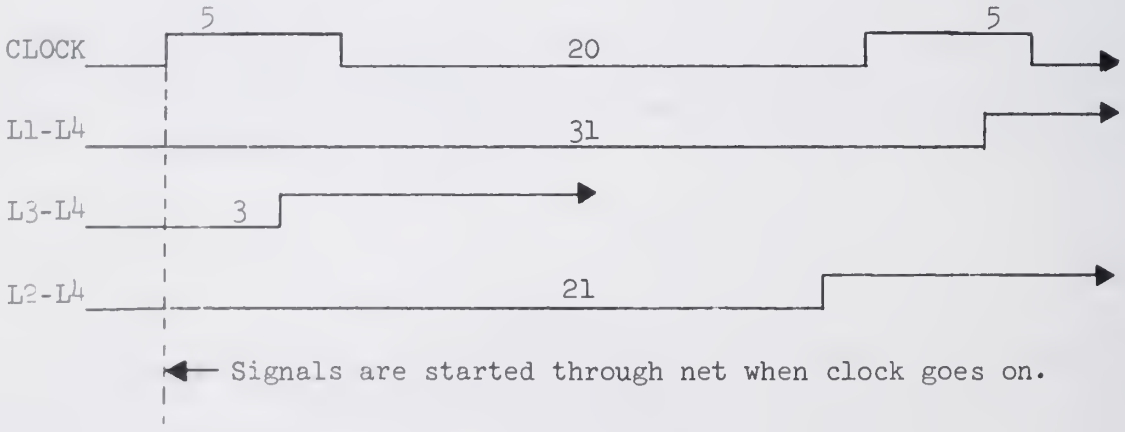
$$D_2 = 5 + 16 = 21$$

Shortest delay from L3 to L4: 2 wires + 1 component

$$D_3 = 2 + 1 = 3$$

Further, suppose that the time between clock pulses is 20 and the length of a pulse is 5 (the units here are not important, so they have been left off). As shown below, the circuit is ill-behaved for two reasons:

1) The time from L1 to L4 is too long, since the signal gets to L4 after the clock pulse to send it on is turned off;
2) The time from L3 to L4 is too short, since the signal gets to L4 during the first clock pulse and thus "runs ahead" of what it should do.



Notice that the shortest path from L2 to L4, as shown above, gets to L4 at the right time. That is, after the pulse that started it on its way and before the next pulse was turned off. For a design to be correct, there must be no nets which exhibit either of the bad traits the above net shows. Deviations will produce errors in the equipment.

For designs which do not use ECL, delay checking is an easier task. Non-ECL logic speeds are approximately 20-30 nsec Compared to wire speeds (the best wiring runs about 2 nsec/ft.) this is relatively slow. So slow, in fact, that in computing the delay for a net, the delay along the wires may be effectively ignored. Algorithms to calculate delay would merely need to count the number of components and multiply by the delay per component. With designs for machines using ECL (such as ILLIAC IV), the problem is not so simple. Since ECL component speeds are about 2 nsec, which is very close to the best wire speeds, delay along wires can no longer be ignored in calculating the dleay for a net. This means that algorithms for delay calculations for designs using ECL will be more complicated than those for non-ECL.

With ECL, counting the components along a path as has been done, will not given an accurate approximation to the delay. For example, a net with few components but long wires may take longer than a net with many components but very short wires.

3. THE DELAY CHECK ALGORITHM

3.1 Development of the Algorithm

The development of the algorithm described in this paper arose out of the need to delay check the board designs of ILLIAC IV. Since ECL is so new, the problems associated with using ECL are also quite new. Because of this, there was no algorithm available to do the job. One had to be developed.

There were many factors which had to be taken into account when deciding what the best algorithm would be. Development time was one of the more important factors. The program had to be working as short a time as possible. Efficiency was considered, but only as a secondary element to time. Therefore, the algorithm had to be simple and easy to convert to a computer language. The language used in writing the program was Burrough's Extended ALGOL. Operations required by the algorithm had to be ones available in that language. Finally, since much of the rest of the ILLIAC IV design automation system already existed, the input data format was fixed. Exactly what data would and would not be available to the delay check program was fixed, and could not be changed. With these contraints in mind, work began to develop the best possible algorithm.

First of all, it was decided that the basic element in the model would be a functional element. This was chosen for its simplicity. It would be much too difficult to break up the component each time a delay calculation had to be made. The components would be broken into functional elements once, and from that point on, everything would be based on functional elements. Now comes the question of how to store all the information in a data structure. Because of storage limitations, information would have to be packed. That is, many pieces of information would be put into one word of memory. This introduces complexity, because packed information must be unpacked to be used. But, since much of the information is binary in nature (a flag indicating true/false, or on/off), the inefficiencies of storing one data bit in a 48-bit computer word could not be tolerated. Along with this it was decided to store all information in one array, each element taking as many words in the array as necessary. This brings up the final consideration. Should the record length for the elements be fixed or variable? Fixed is simpler and easier to work with but wastes a lot of space. Because boards are very large (as many as 1000 functional elements), and because some elements require much more space for information than others, it was decided to use variable length records. Each record would start on a word boundary but could be any number of words long.

The record for an element contains backward pointers to all elements which have sources which feed this element directly, forward pointers to all elements having loads for the sources in this element, and other information about the element; such as, type, number of reference (forward and backward), and delay information. Using these pointers, the program is able to follow signals through the nets, adding up wire and component delays as it goes from latch to latch. By doing this, maximum and minimum delay times can be calculated for all nets.

The data structure consists of variable length records of packed data. Internal pointers allow for tracing of nets both forward and backward in the circuit. (The actual format of these structures is given in Appendix 2.)

3.2 General Information

If a program is to be written for a computer to perform the delay check function, an algorithm must be devised which models each net internally so that the delay can be calculated using both component and wire delays.

This paper presents such an algorithm and describes a program which was written based on the algorithm.

The algorithm consists of two parts:

1) The compiler, which creates the internal model of all the nets, and
2) The propagator, which uses the model to propagate the delays along the nets to find total delay from latch to latch.

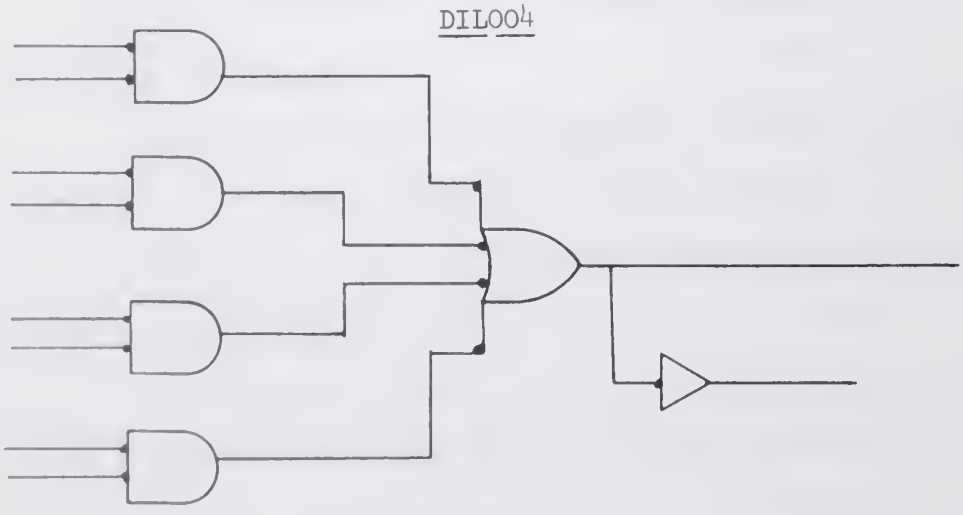
3.3 The Compiler

The task of the compiler, as stated above, is to generate the internal model of the nets. The model was designed with the idea in mind that the input data would contain the following information about every pin in the net.

1) The signal name on the wire connected to this pin.
2) The component name or connector pin name associated with this pin.
3) What type of component this pin is on (or if this is a connector pin, it indicates this).
4) Which pin on the component this is.
5) Whether this pin represents a source or a load to the signal.
6) The delay along the wire from the source to each of its loads.

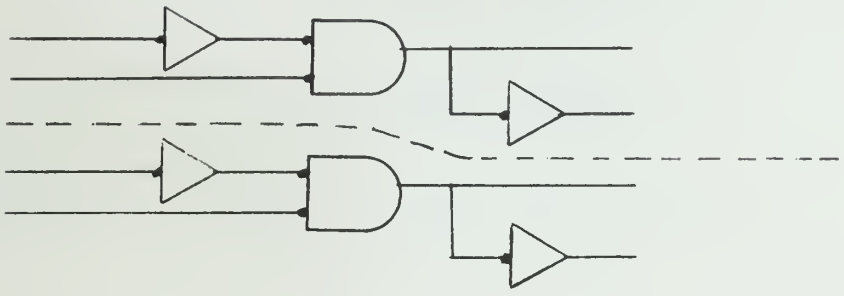
The model consists of several lists: one main information net list and several peripheral supporting lists. The main list contains one record for each functional element in the net. A given component package may be a functional element in itself or it may contain several functional elements. The criterion for defining a functional element is based on its output. All outputs from a single functional element must come directly from the same gate within one component. If the output comes from a gate, then through a NOT, and then out of the component, this is also taken to mean directly from the gate. Shown below are four ILLIAC IV component packages with explanations of the functional element breakdown in each.

DIL004



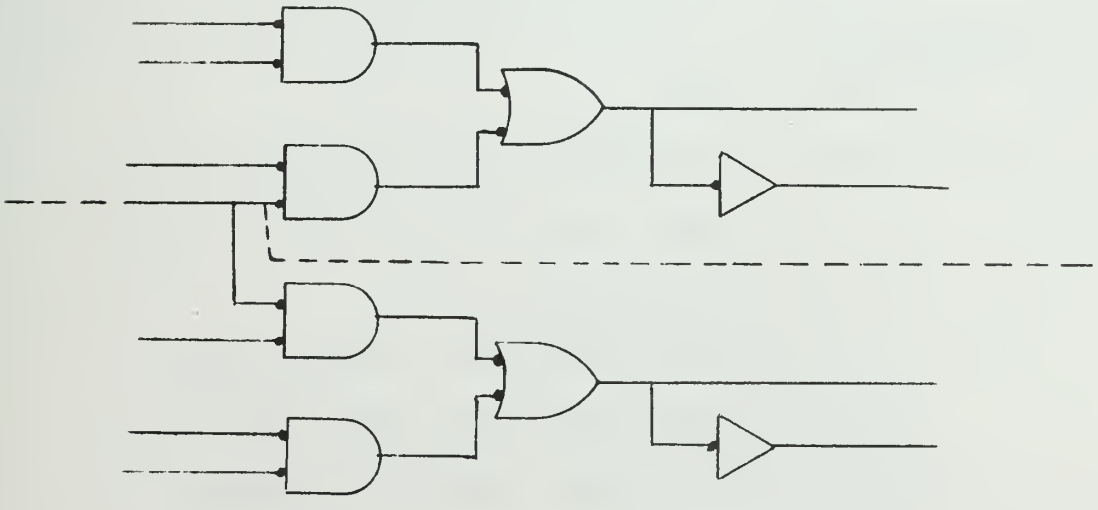
In this package, since both outputs come from the same gate (one through a NOT, the other straight out), the entire package is one functional element.

DILDC



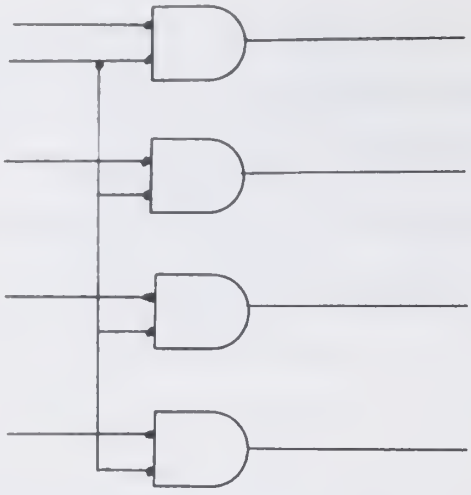
In this package there are two functional elements (separated by broken line). Each has two outputs, one the actual value of the element, the other the NOT of this value.

DILO11



In this package there are also two functional elements. Notice that they share one of the inputs.

DIL007



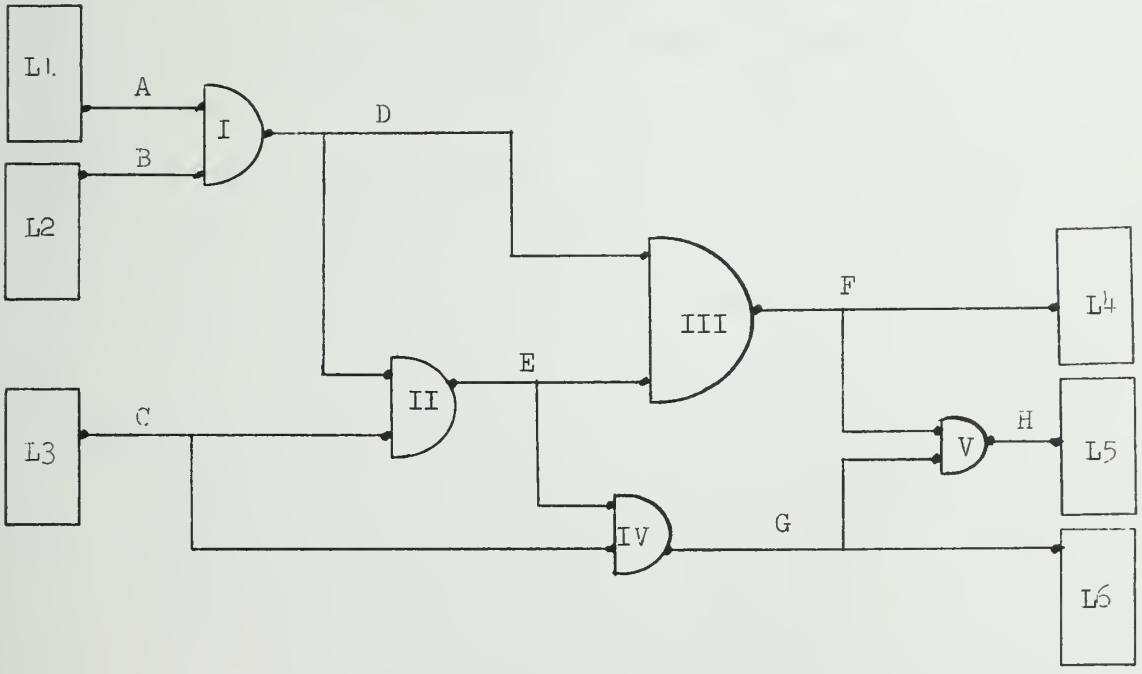
In this package there are four functional elements, each sharing one common input and having but a single output.

The records are of variable length and contain the following information about the functional element:

1) Is this a latch?
2) Is this a connector pin and, if so, a pointer to a special connector pin list, and is this an input or output connector pin?
3) If this is not a connector pin, how many input signals are there and how many output signals?
4) Pointers for each input signal to the element that the source is on.
5) The wire delay from the source to the load for each input.

6) Pointers for each output signal to the elements to which they go.

Shown below is a partial net, together with the main information net list that the compiler would generate for it. The net is drawn at the functional element level for simplicity and the compiler puts four pieces of information in each word in the list.



	WORD ADDRESS	INFORMATION
(L1)	0:	latch, not connector pin, 0 inputs, 1 output
	1:	6 (Pointer to Element I) , , ,
(L2)	2:	latch, not connector pin, 0 inputs, 1 output
	3:	6 (Pointer to Element I) , , ,
(L3)	4:	latch, not connector pin, 0 inputs, 2 outputs
	5:	9 (Ptr. to II), 15 (Ptr. to IV) , ,
(I)	6:	not latch, not connector pin, 2 inputs, 2 outputs
	7:	0 (Ptr. to L1), 2 (Ptr. to L2), Delay from L1 Delay from L2
	8:	9 (Ptr. to II), 12 (Ptr. to III), ,
(II)	9:	not latch, not connector pin, 2 inputs, 2 outputs
	10:	6 (Ptr. to I), 4 (Ptr. to L3), Delay from I, Delay from L3
	11:	12 (Ptr. to III), 15 (Ptr. to IV), ,
(III)	12:	not latch, not connector pin, 2 inputs, 2 outputs
	13:	6 (Ptr. to I), 9 (Ptr. to II), Delay from I, Delay from II
	14:	18 (Ptr. to V), 21 (Ptr. to L4), ,
(IV)	15:	not latch, not connector pin, 2 inputs, 2 outputs
	16:	9 (Ptr. to II), 4 (Ptr. to L3), Delay from II, Delay from L3
	17:	18 (Ptr. to V, 25 (Ptr. to L6), ,
(V)	18:	not latch, not connector pin, 2 inputs, 1 output
	19:	12 (Ptr. to III), 15 (Ptr. to IV), Delay from III, Delay from IV
	20:	23 (Ptr. to L5), , ,

	WORD ADDRESS	INFORMATION
(L4)	21:	latch, not connector pin, 1 input, 0 outputs
	22:	12 (Ptr. to III), , ,
(L5)	23:	latch, not connector pin, 1 input, 0 outputs
	24:	18 (Ptr. to V), , ,
(L6)	25:	latch, not connector pin, 1 input, 0 outputs
	25:	15 (Ptr. to IV), , ,

Using the above list, the program can start at latches L1, L2, and L3, and by following pointers go from element to element until it reaches L4, L5, and L6. As the program goes along it keeps track of the delay along each path. It will know when it gets to each of L4, L5, and L6 what the corresponding delays are.

Notice that signal names (shown as capital letters) completely disappear in the information list.

The input to the compiler would then need to be one record for each "pin" in the design. The pins in the example are shown as small black dots. Thus, in the above example, the input would consist of only 17 records. Each record would contain the signal name, logic component name, a flag indicating whether this is the source or a load for the signal, if this is a load the "wire delay" from the source, and the type of element this pin is on.

The compiler takes the input records and first sorts by signal name. The source for each signal is distinguished from its loads and pointers are formed from the source to each load and from

each load to the source. The updated records (with pointers included) are now sorted by logic element name. Now, by extracting the pointers for each signal and discarding signal names, the list shown above can be generated.

It should be noted that in an actual application of this algorithm (as in the one described later in this paper), the process can be made more complicated by allowing more complicated components than used above. The basic algorithm, however, would still be the same.

3.4 The Delay Propagator

The propagator is very simple, given the list generated by the compiler. All that need be done is that a copy of the list be made for each of two computations. Both the maximum delay time and the minimum delay time from latch to latch must be calculated. If each calculation begins at input connector pins (a list of which can be easily generated by the compiler) and follows each through the list until output connector pins are reached. At this point all delays will be known and output of this information can be generated. The output would describe the delay at each latch from the previous latches.

For nets which start on a latch on one board and reach the next latch on a different board, special interfacing must be provided for. This is merely a data handling problem and will not be discussed in this paper.

4. THE DELAY CHECK PROGRAM

The program following the above described algorithm was written to run on a B5500 in Burrough's Extended ALGOL. It is intended to run after the Post-Processor portion of the design automation package furnished by Burroughs to aid in the design of ILLIAC IV. For this reason the program is, in some places, more specifically orientated to the problem of designing ILLIAC IV than a general delay checking program should be.

The compiler takes its primary input from a file generated by the Post-Processor. This is the Extended Pin List File (PLF). This file consists of card image records and should contain all information for one board and no more. Each record represents one pin on the board as is formatted as follows:

col. 9-14	Package type
col. 16-19	Component identification
col. 21-23	Pin number
col. 25-36	Signal name
col. 38	Source/load key
col. 65-68	Delay (on source pins only)

The package type is a six-character alphanumeric code which designates on which of the several types of modules known to the program this pin is. A list of all such modules should be provided as explained in Appendix 1. Among the possible codes should be one for connector pins and one for latches. If the code is not one of the allowable codes, the program will print an appropriate error message and terminate.

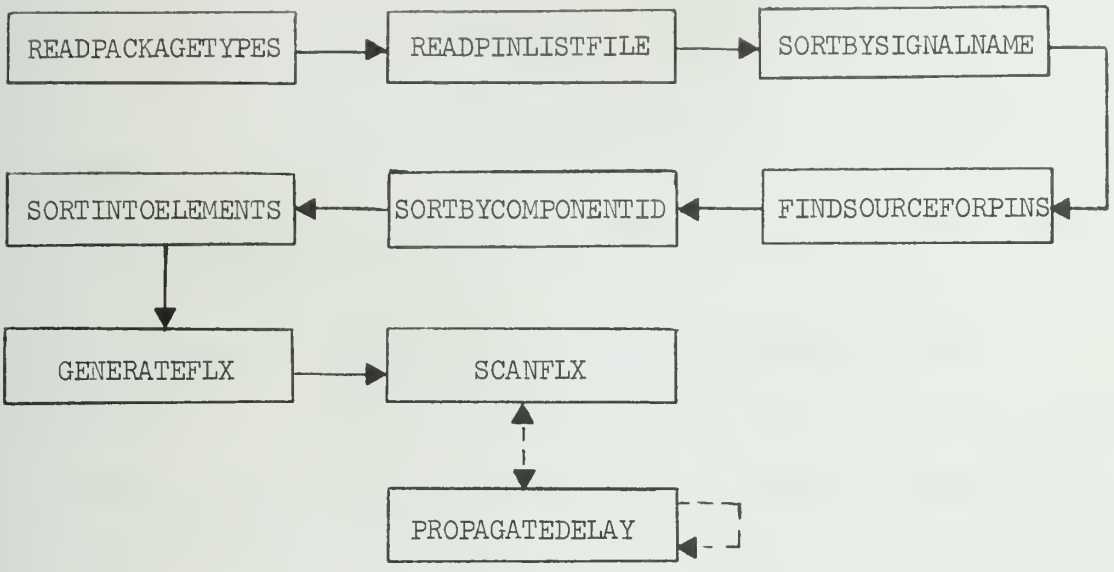
The component ID is a unique four-character alphanumeric name given to each component on the board. It is used to group the pins by component for processing by the program. Since this name can be any four-character string, no check is made for correctness and all names are ass ed to be correct. One exception to this is that connector pins will have as their component ID a name of the type: P-xx where xx is a number from 00 to 99.

The signalname is a twelve-character alphanumeric name, unique for each signal in the system. The signalname will be the name associated with the delay on all output connector pins.

The source/load key is the character S or L. As would be expected, the S indicates that this is a source pin and the L indicates a load.

The delay is the amount of wire delay on each source signal. This is the total delay for the signal from the source to the furthest load downstream. It was decided that, even though it is not absolutely correct, this delay would be used as the delay time from the source to each load. Since there is only one delay associated with each source, it will be present only on records which are marked as sources.

The program is written as a series of sequential program segments. Each one performs some transofrmation on the data files until the input has been completely changed into the form necessary for delay propagation. A general flow diagram for these segments, along with a short description of what each does, follows.



READPACKAGETYPES reads, from file PACKAGE, card image records which describe the package types allowed on the board being delay checked. The record also will contain the number of pins in the package and the name, or pin ID, of each pin.

READPINLISTFILE reads card image records from file PINLISTFILE. Each record contains information for one pin on the board. This segment checks for errors in package types and pin names.

SORTBYSIGNALNAME not only sorts by signal name but also puts the source(s) for each signal ahead of the loads.

FINDSOURCEFORPINS puts a pointer in GREX (see Appendix 2) for each load to its source; and for each source puts the number of loads on that source.

SORTBYCOMPONENTID does a pseudo-sort by component name, source or load, and pin number. The arrays COMPI and COMPJ are formed to contain the correct order of GREX as though it were sorted. The sort is done first by component name, then by putting sources ahead of loads, and finally by putting the pin numbers in numerical order.

SORTINTOELEMENTS separates the element(s) in each component, keeping a count of them for the entire board. It generates an array called COMPONENT (see Appendix 2) with one entry for each component. Each entry contains the package type and lists the element number(s) of the element(s) in this component.

GENERATEFLX generates the FLX array (see Appendix 2) which is the complete model of the board. This is the array which is used to propagate the delay.

SCANFLX initiates delay propagation and calls PROPAGATEDELAY to do the actual calculations. This is the part which must be executed once for maximum delay(s) and once for minimum delay(s).

PROPAGATEDELAY is a recursive procedure which calculates the delay from one element to the next by following pointers through the FLX array.

Detailed flowcharts of the program are in Appendix 3 for those interested; and Appendix 2 describes, in detail, the format of the arrays used by the program to build the internal model of the board.

APPENDIX 1

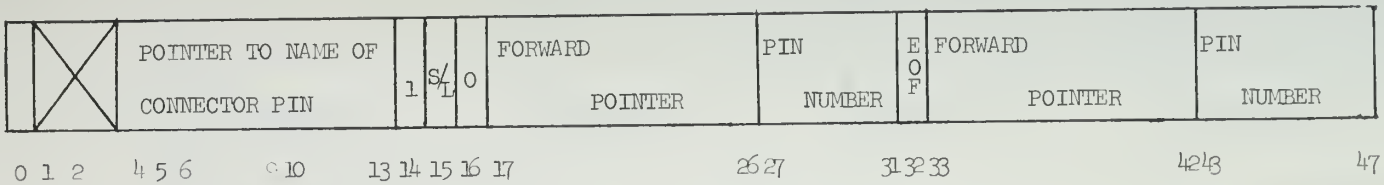
FORMAT OF PACKAGE TYPE INPUT

First card:	cols 1-4	Number of types - 1
	cols 5-10	Connector Pin Code
	cols 11-16	Latch Code
Type cards:	cols 1-6	Type Code
	col 8	Number of elements in this package
	cols 9-10	Number of pins in this package - 1
Repeat this 5 column field until all pins are defined (this may take two cards).	cols 11-13	Pin name
	cols 14-15	Pin position on package

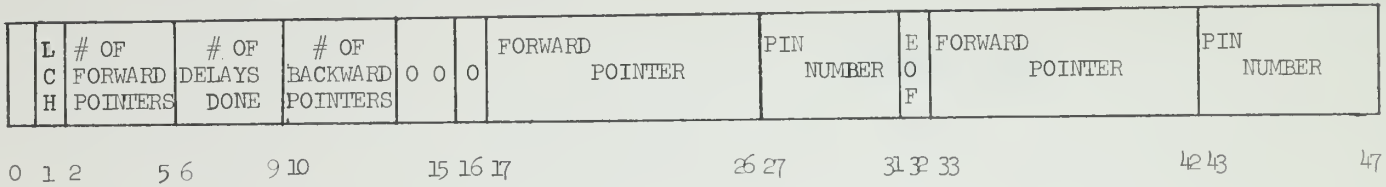
APPENDIX 2

DESCRIPTIONS OF THE FORMAT OF THE GREX, FLX, and COMPONENT ARRAYS

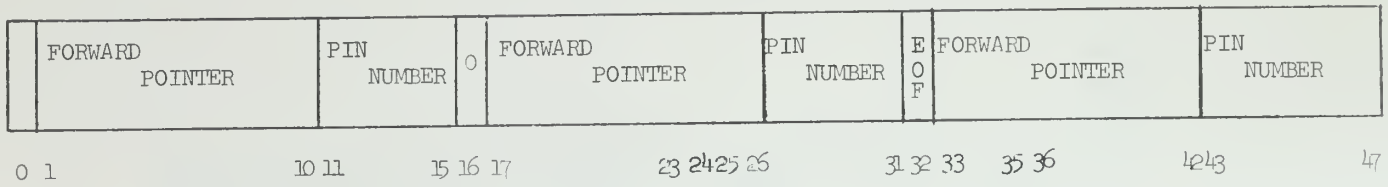
FLX ARRAY IN ITS 5 FORMS



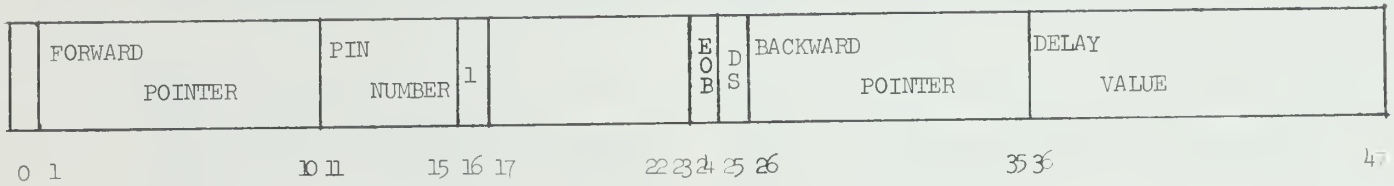
FIRST WORD OF ELEMENT IF ELEMENT IS A CONNECTOR PIN



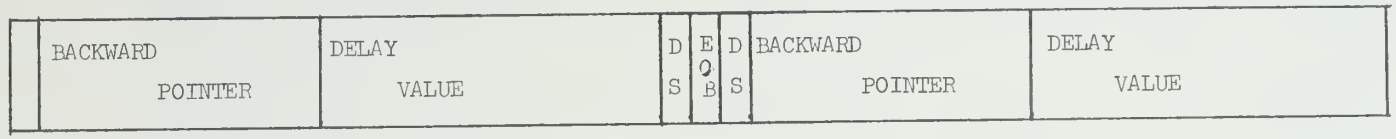
FIRST WORD OF ELEMENT IF ELEMENT IS NOT A CONNECTOR PIN



SECOND, THIRD, ... AS NEEDED FOR FORWARD POINTERS (3 PER WORD)



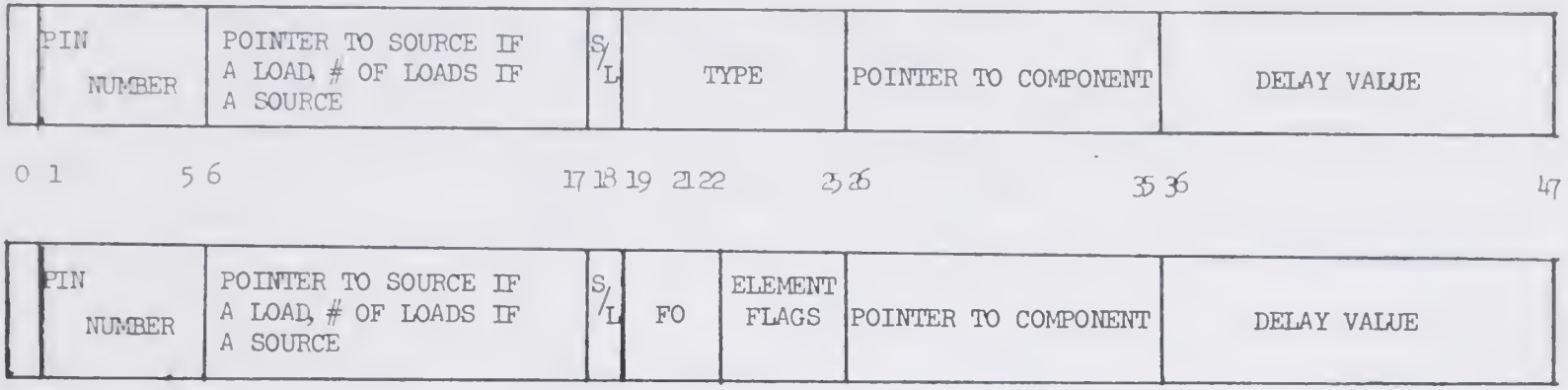
SPLIT WORD - ONE FORWARD POINTER AND ONE BACKWARD POINTER



FINAL WORDS HAVE ONLY BACKWARD POINTERS

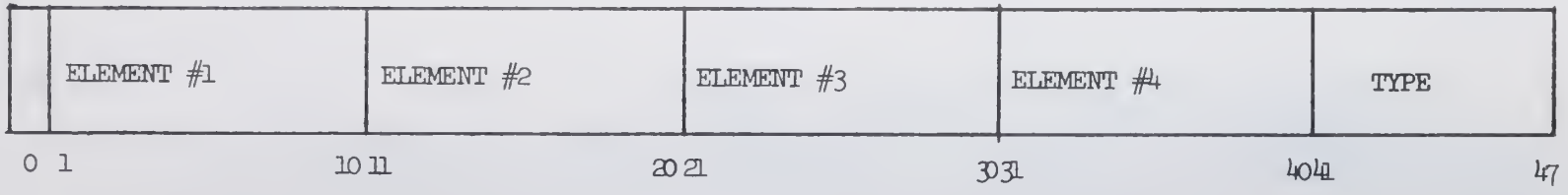
Abbreviations:
- S/L 0 = SOURCE; 1 = LOAD
- EOF = 1 INDICATES THAT THIS IS THE LAST WORD OF FORWARD POINTERS
- EOB = 1 INDICATES THAT THIS IS THE LAST WORD OF BACKWARD POINTERS
- DS = 1 INDICATES THAT DELAY HAS BEEN PROPAGATED TO THIS POINT
- LCH = 1 INDICATES THAT THIS IS A LATCH

GREX ARRAY IN 2 FORMS



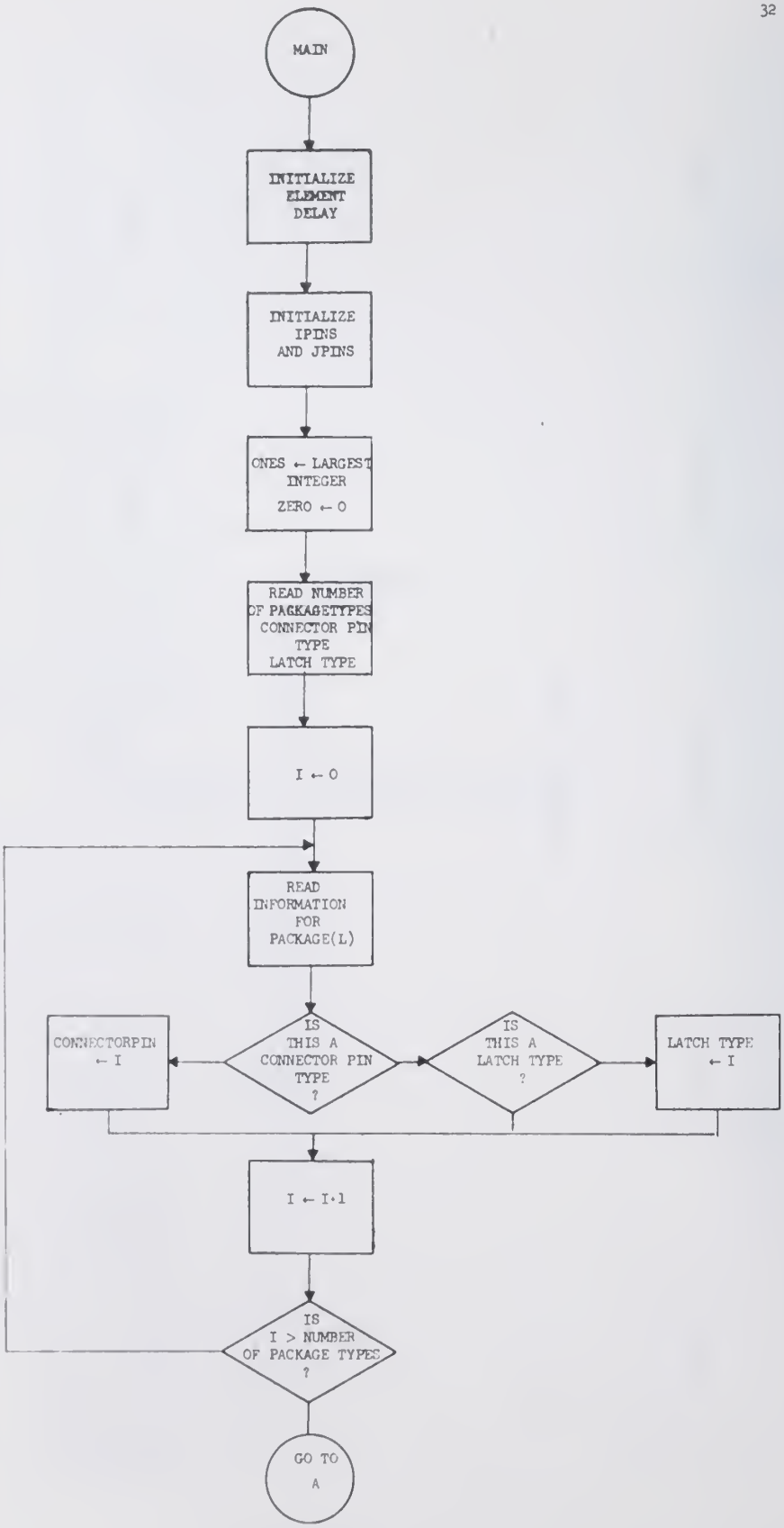
Abbreviations: S/L 0 = SOURCE; 1 = LOAD
FO = 111 INDICATES FIRST WORD OF COMPONENT

COMPONENT ARRAY

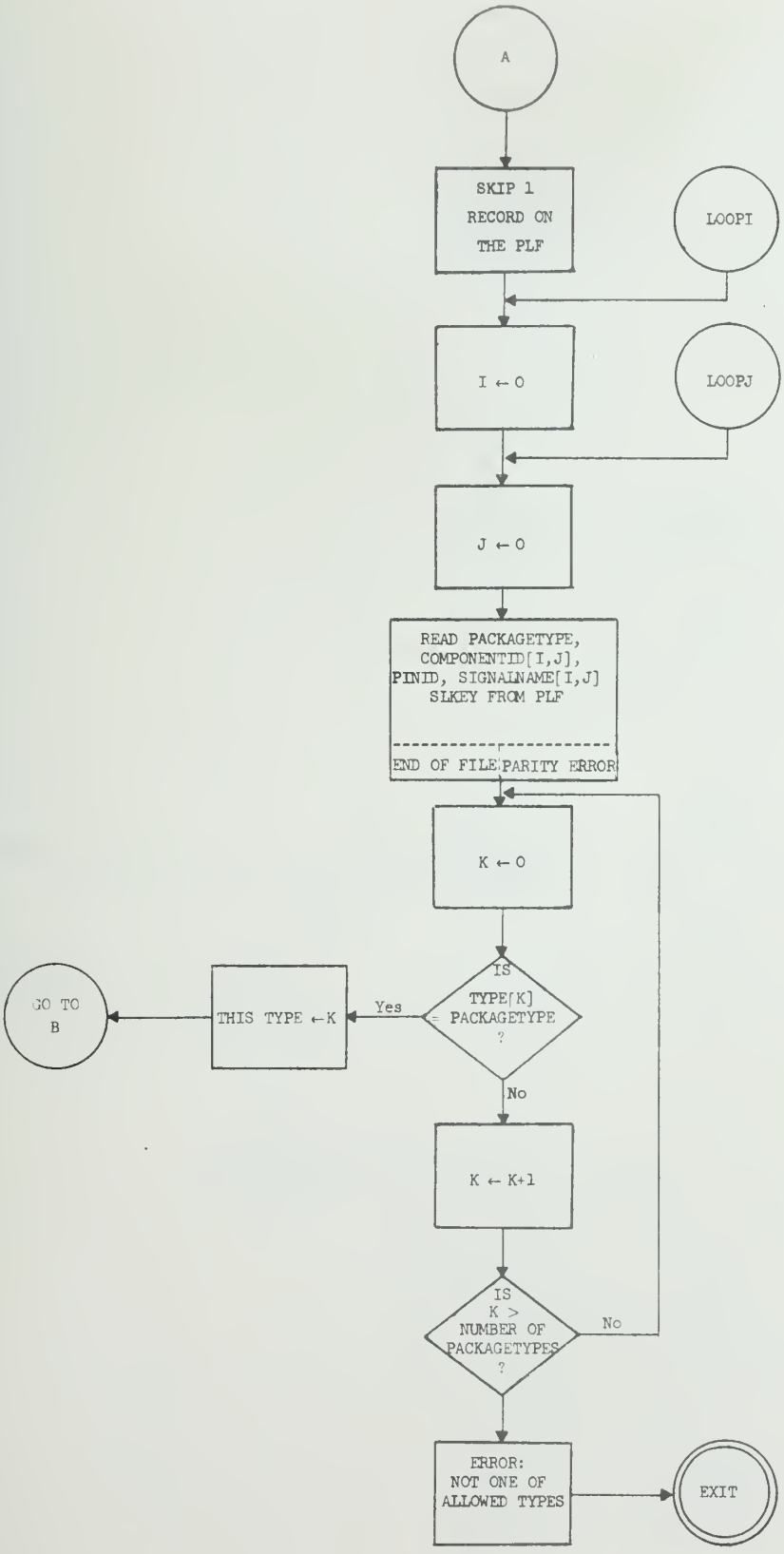


APPENDIX 3

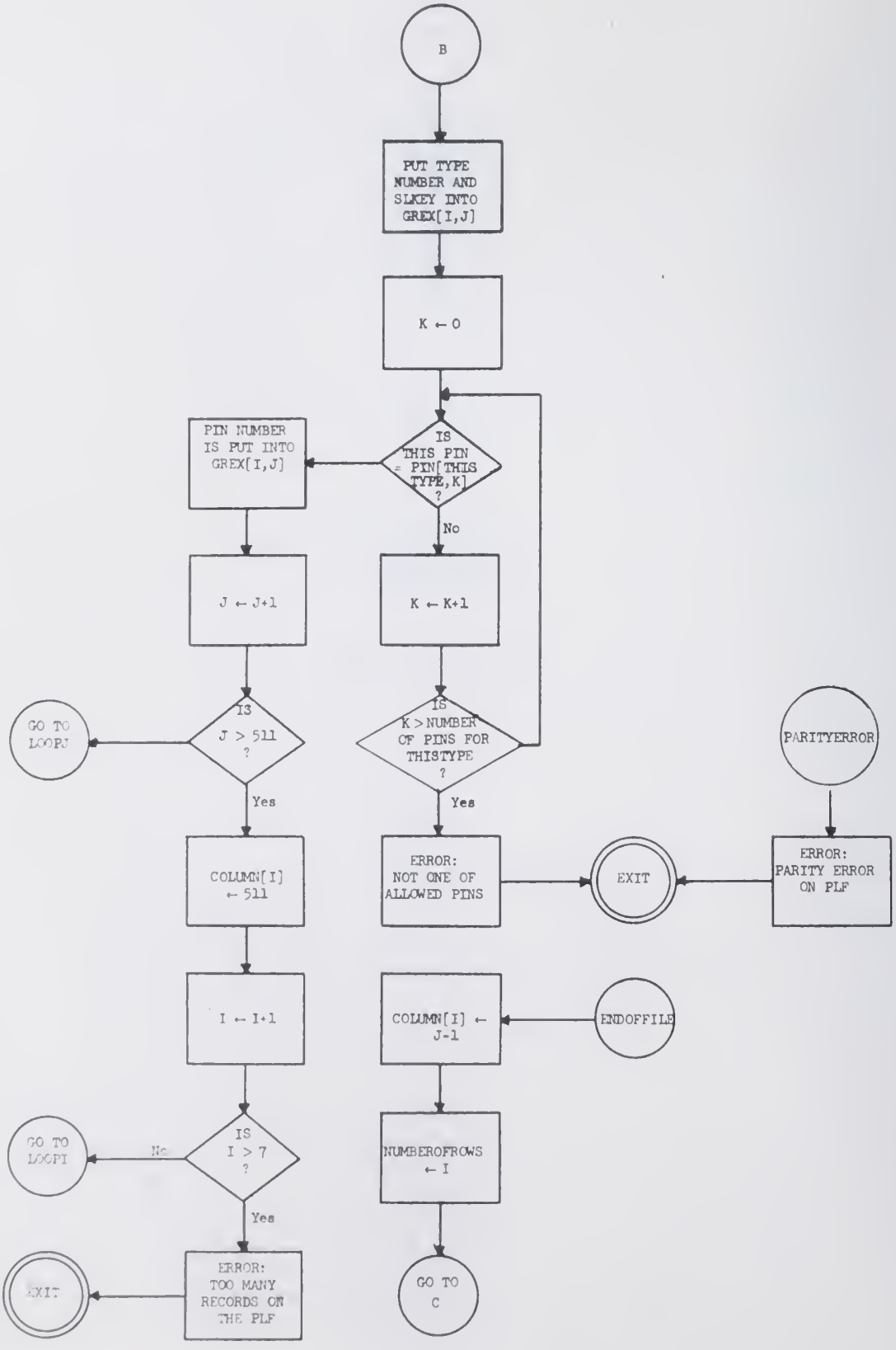
DETAILED FLOWCHARTS OF PROGRAM

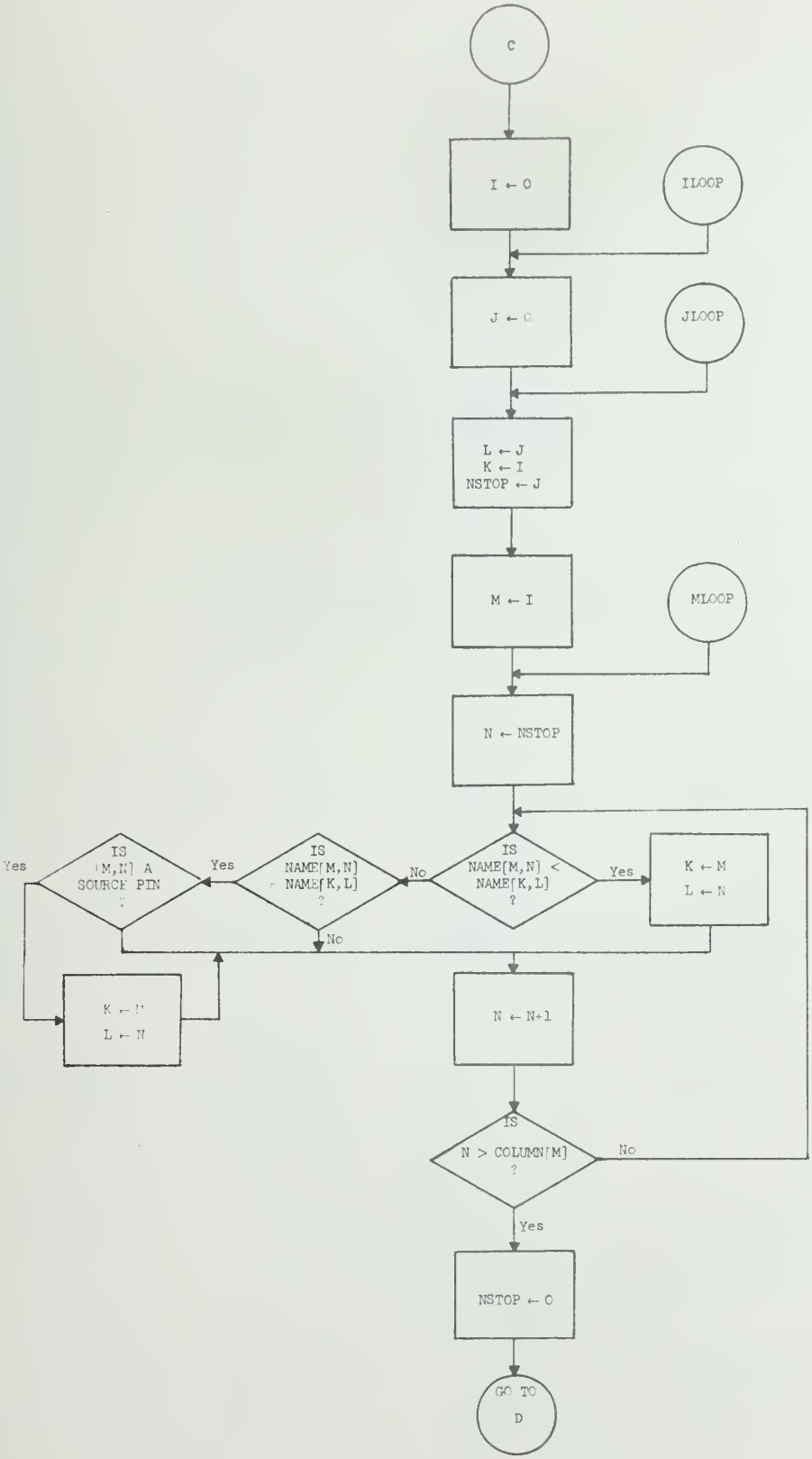


READPACKAGETYPES

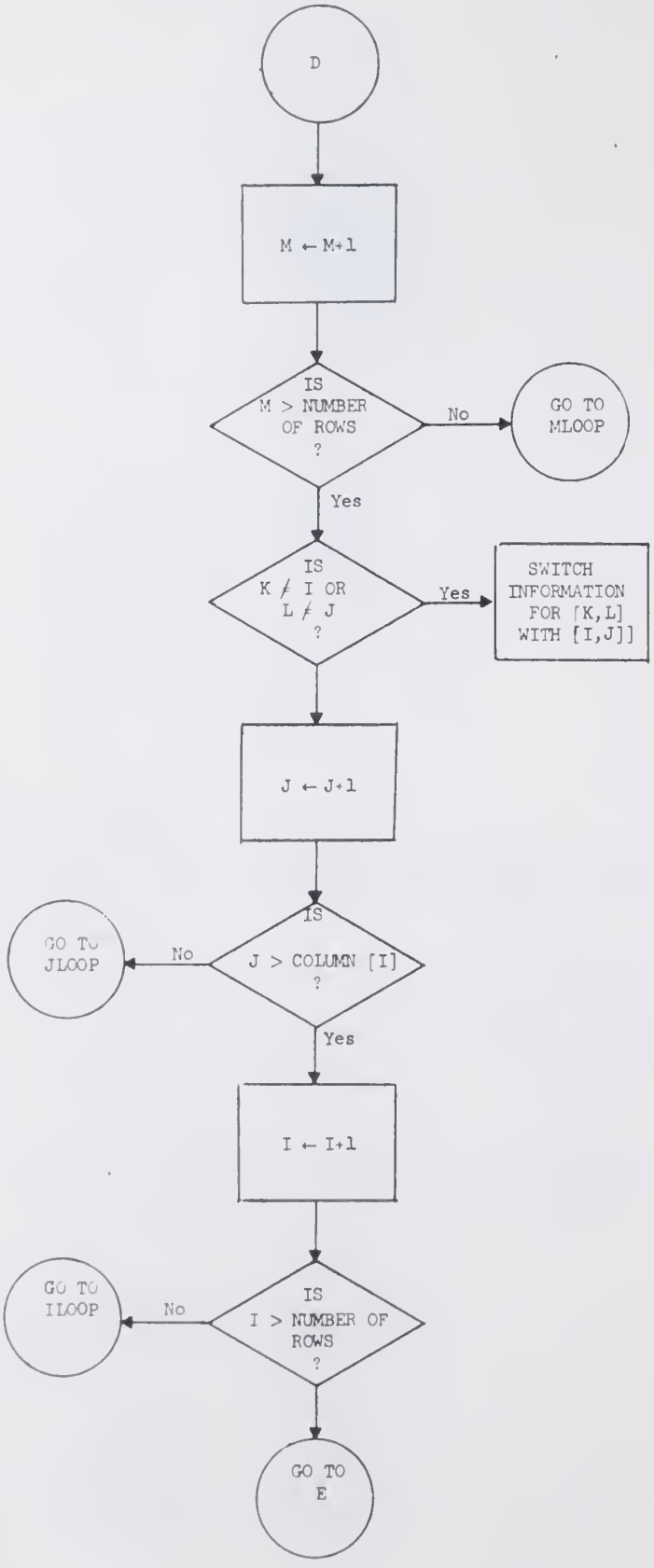


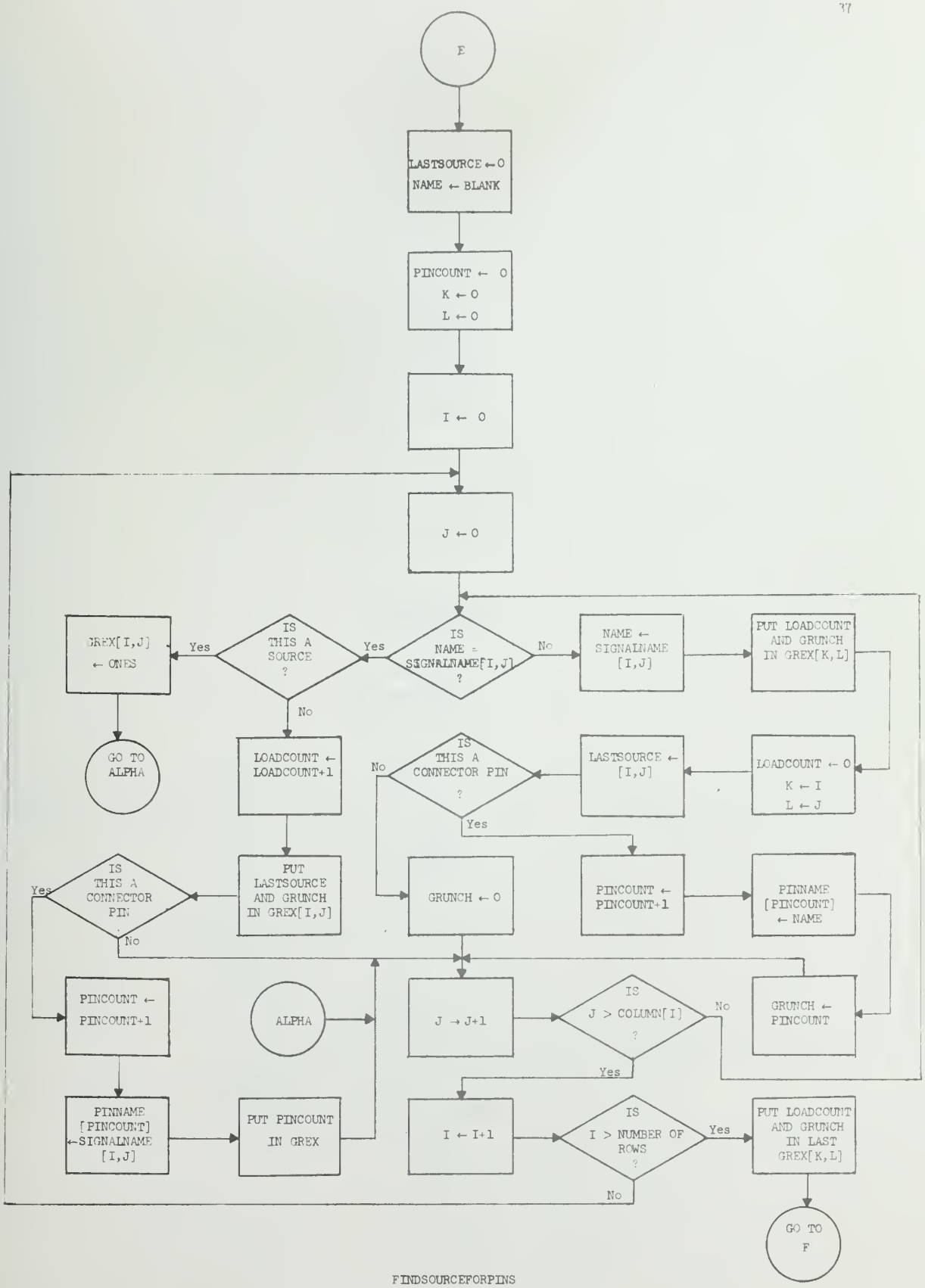
READPINLISTFILE


B
PUT TYPE NUMBER AND SLKEY INTO GREX[I,J]
K ← 0
IS THIS PIN = PIN[THIS TYPE,K] ?
PIN NUMBER IS PUT INTO GREX[I,J]
No
J ← J+1
K ← K+1
IS J > 511 ?
GO TO LOOPJ
IS K > NUMBER OF PINS FOR THISTYPE ?
Yes
Yes
COLUMN[I] ← 511
ERROR: NOT ONE OF ALLOWED PINS
EXIT
PARITYERROR
ERROR: PARITY ERROR ON PLF
I ← I+1
COLUMN[I] ← J-1
ENDOFFILE
IS I > 7 ?
GO TO LOOPI
No
NUMBEROFROWS ← I
Yes
EXIT
ERROR: TOO MANY RECORDS ON THE PLF
GO TO C

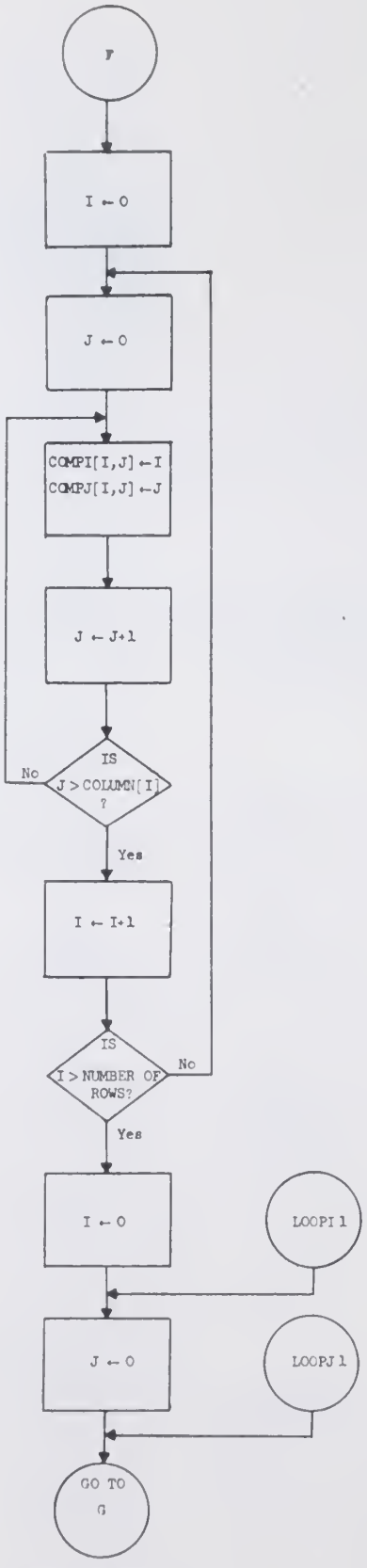


SORTBYSIGNALNAME

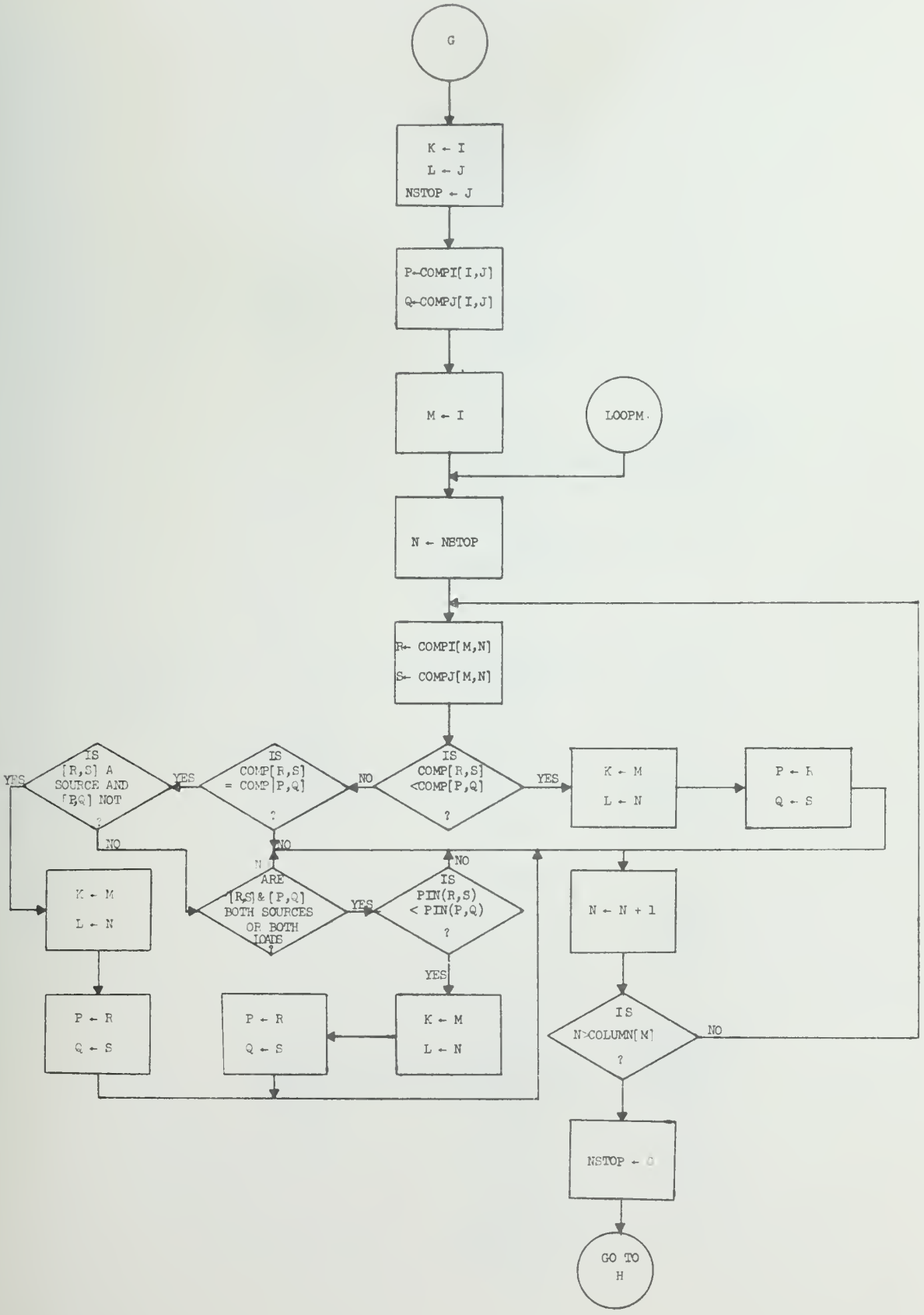

D
M ← M+1
IS M > NUMBER OF ROWS ?
No
GO TO MLOOP
Yes
IS K ≠ I OR L ≠ J ?
Yes
SWITCH INFORMATION FOR [K,L] WITH [I,J]]
J ← J+1
IS J > COLUMN [I] ?
No
GO TO JLOOP
Yes
I ← I+1
IS I > NUMBER OF ROWS ?
No
GO TO ILOOP
GO TO E

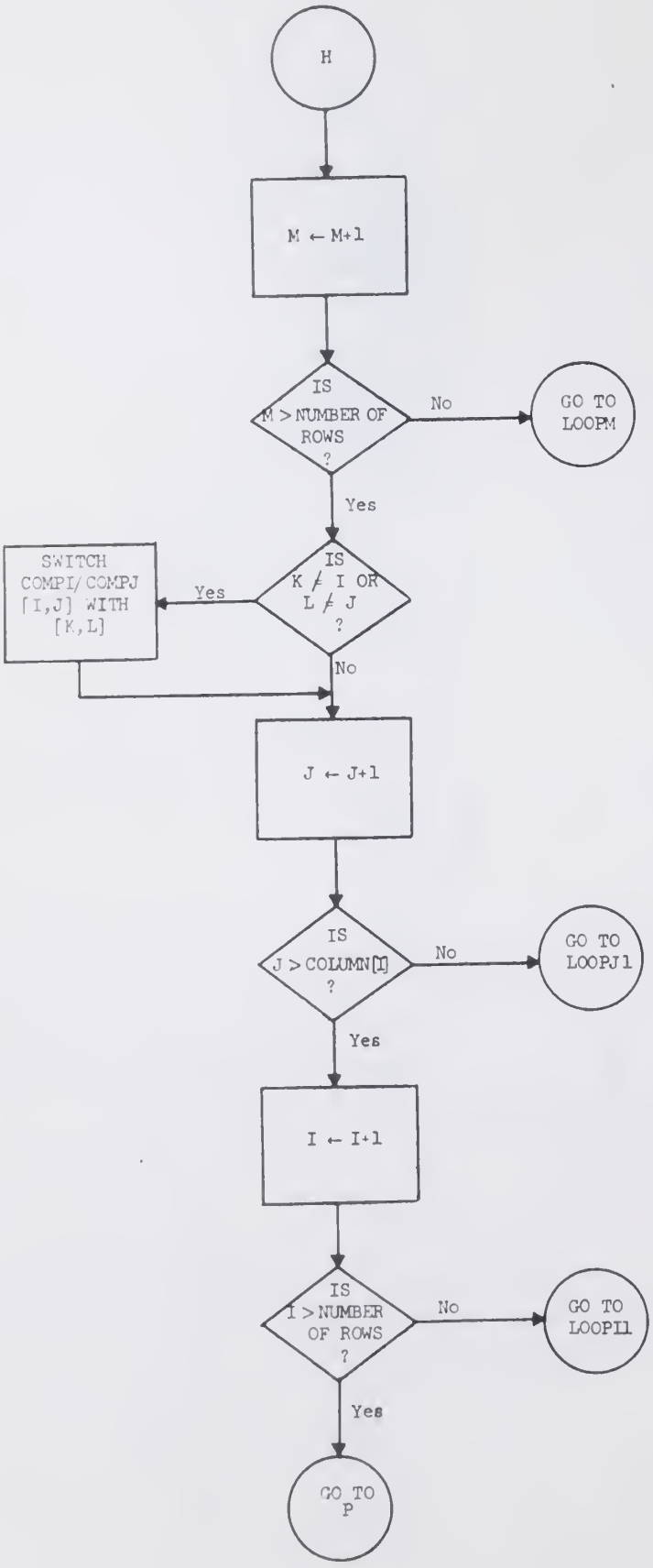


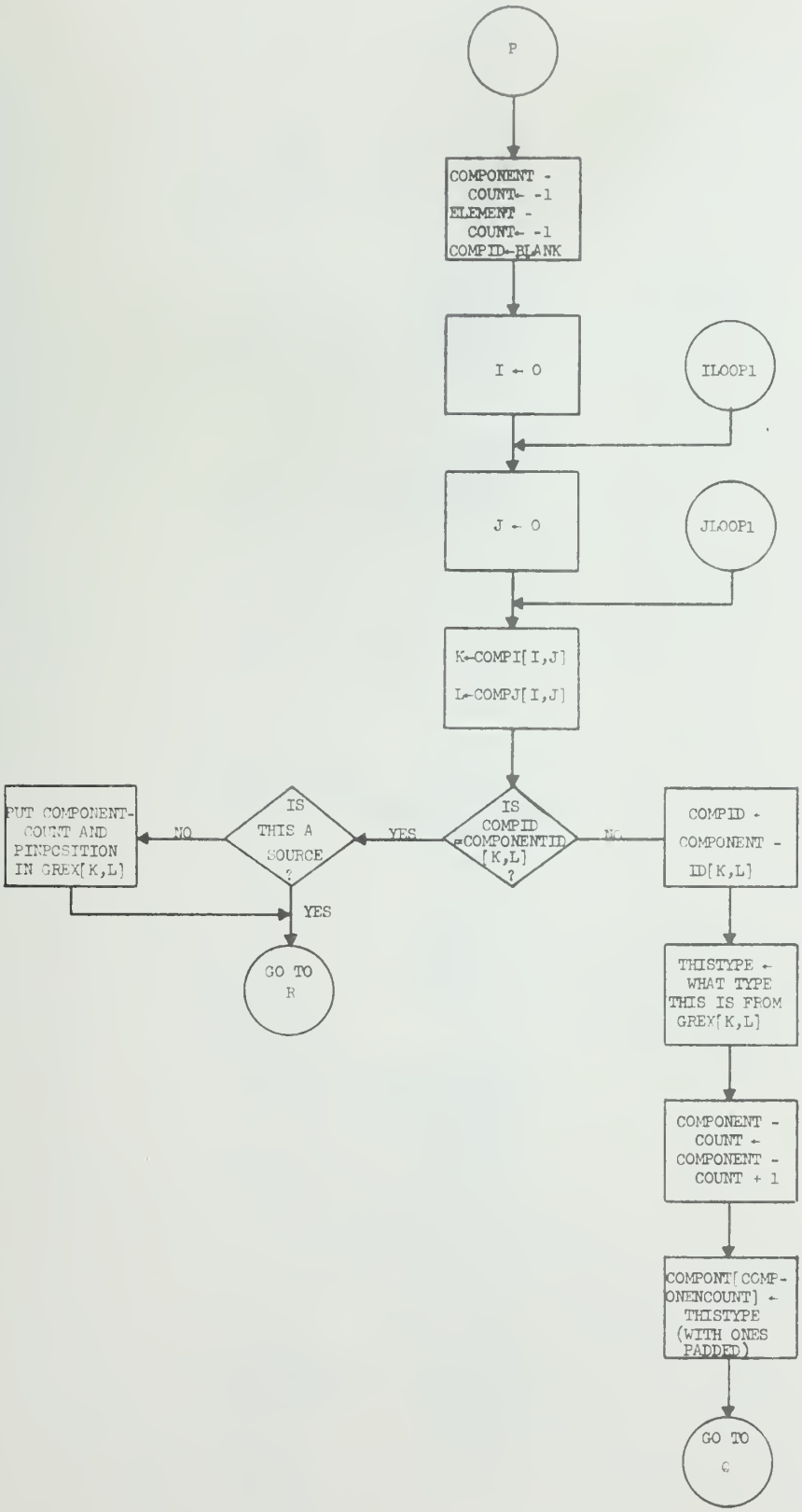
FINDSOURCEFORPINS



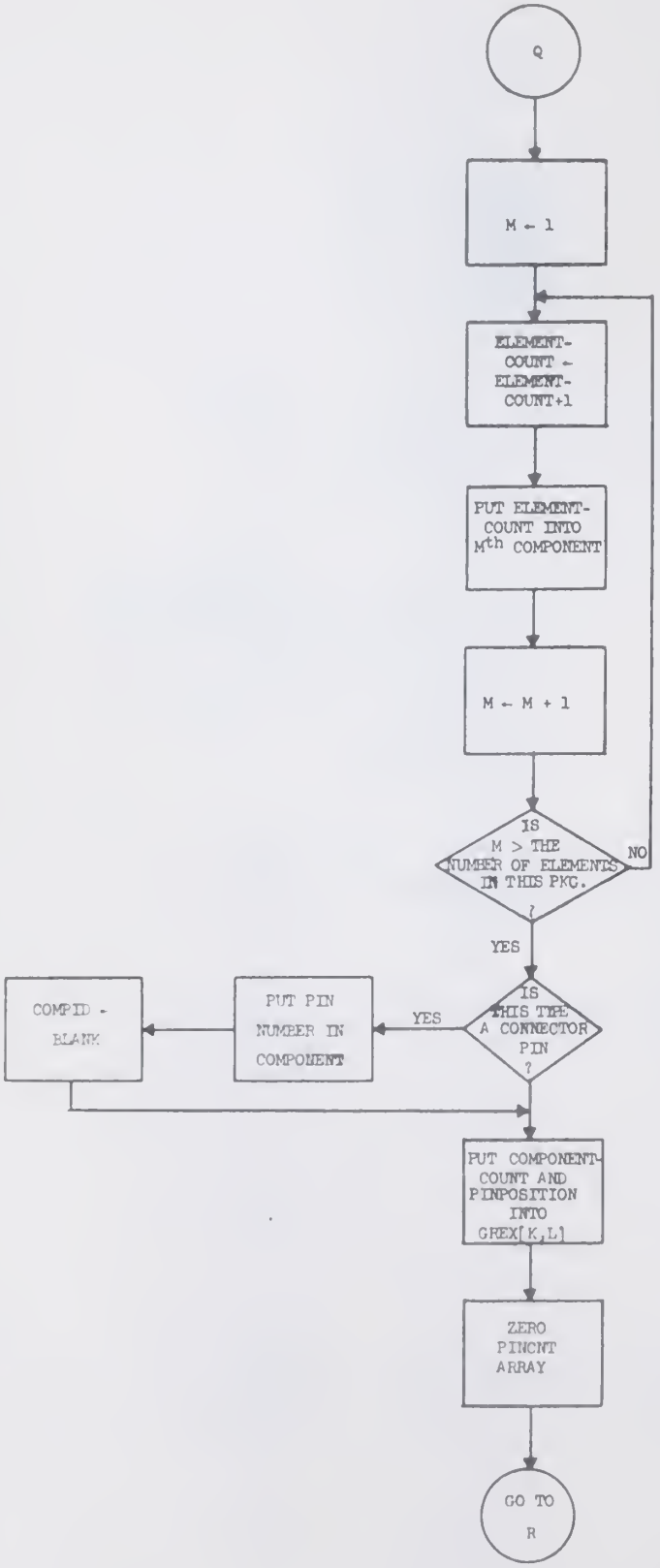
SORTBYCOMPONENTID

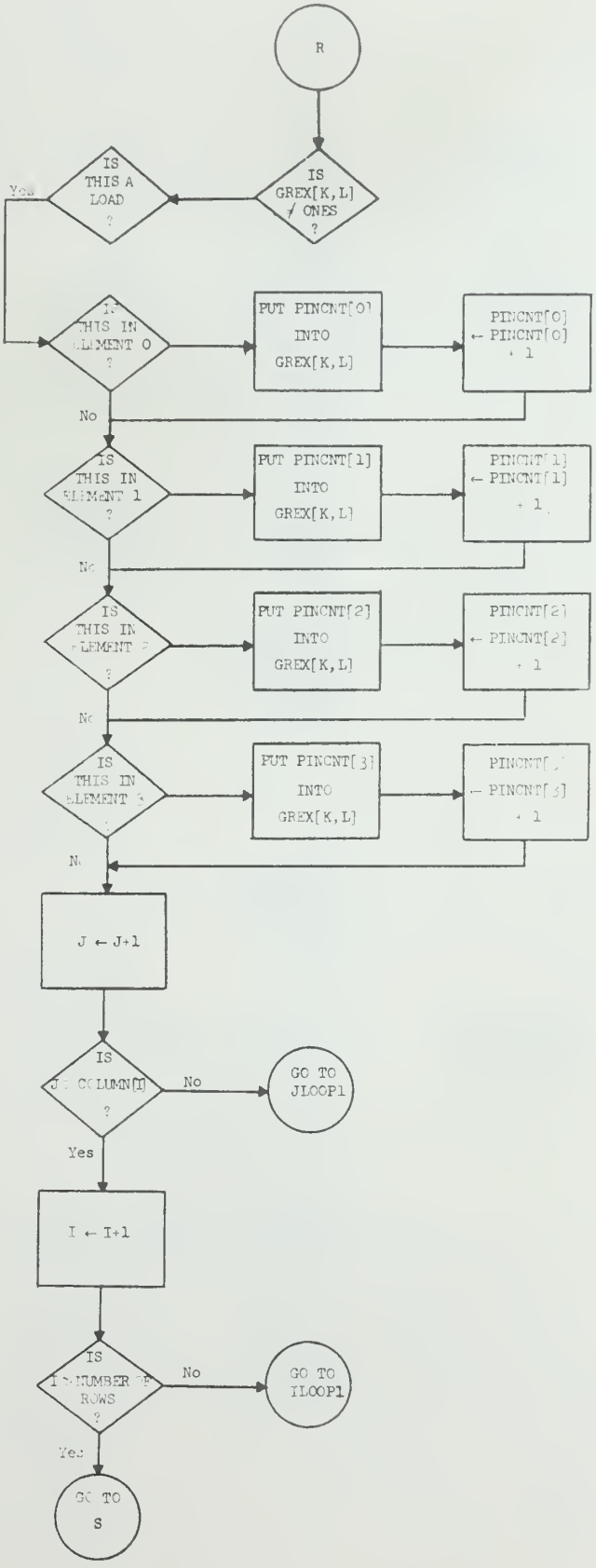

G
K ← I
L ← J
NSTOP ← J
P←COMPI[I,J]
Q←COMPJ[I,J]
M ← I
LOOPM
N ← NSTOP
R← COMPI[M,N]
S← COMPJ[M,N]
IS [R,S] A SOURCE AND [P,Q] NOT ?
IS COMP[R,S] = COMP[P,Q] ?
IS COMP[R,S] <COMP[P,Q] ?
K ← M
L ← N
P ← R
Q ← S
YES
NO
ARE [R,S]&[P,Q] BOTH SOURCES OR BOTH LOADS ?
IS PIN(R,S) < PIN(P,Q) ?
N ← N + 1
IS N>COLUMN[M] ?
NSTOP ← 0
GO TO H

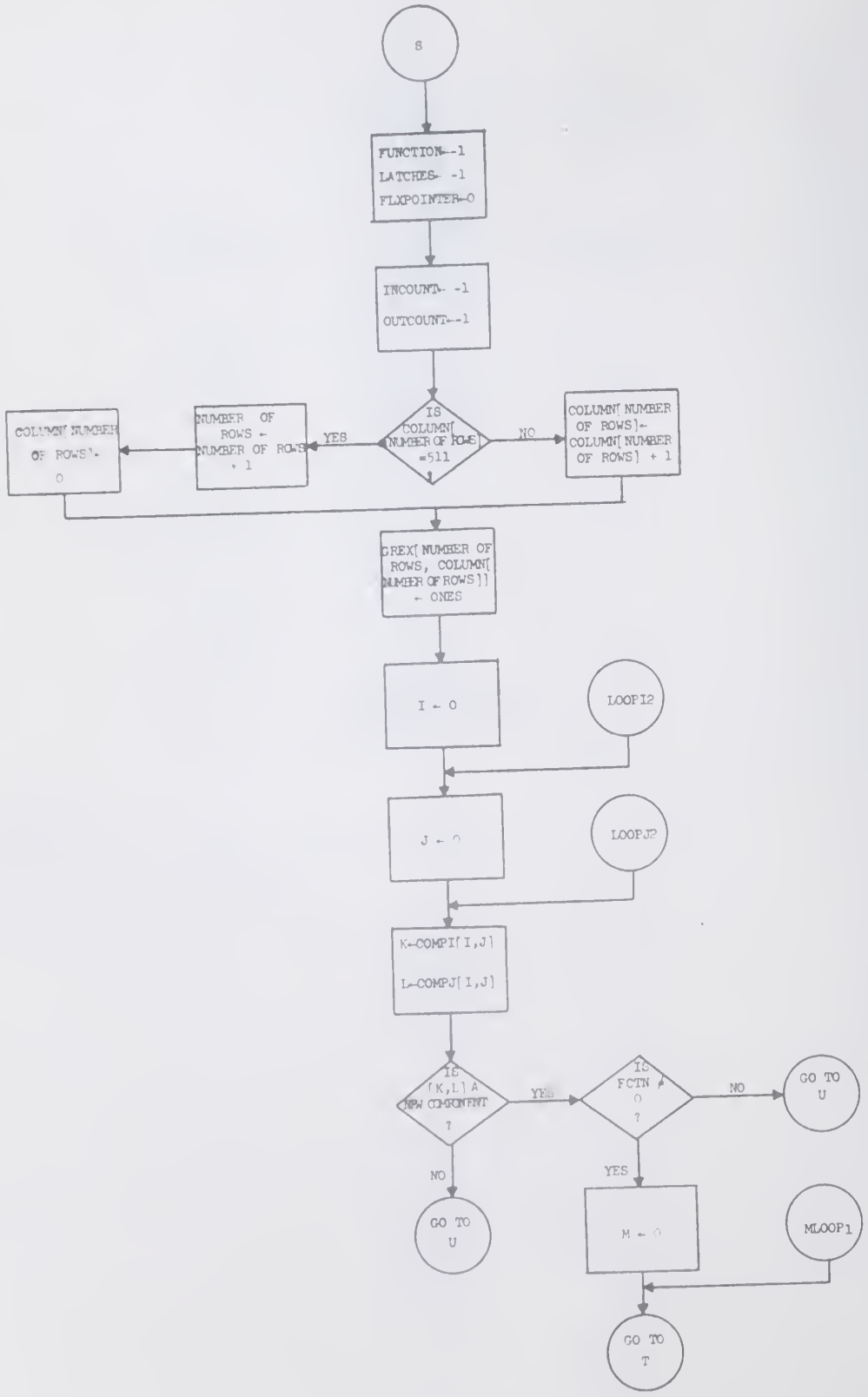

H
M ← M+1
IS M > NUMBER OF ROWS ?
No
GO TO LOOPM
Yes
IS K ≠ I OR L ≠ J ?
Yes
SWITCH COMPI/COMPJ [I,J] WITH [K,L]
No
J ← J+1
IS J > COLUMN[I] ?
No
GO TO LOOPJ1
Yes
I ← I+1
IS I > NUMBER OF ROWS ?
No
GO TO LOOPI1
Yes
GO TO P



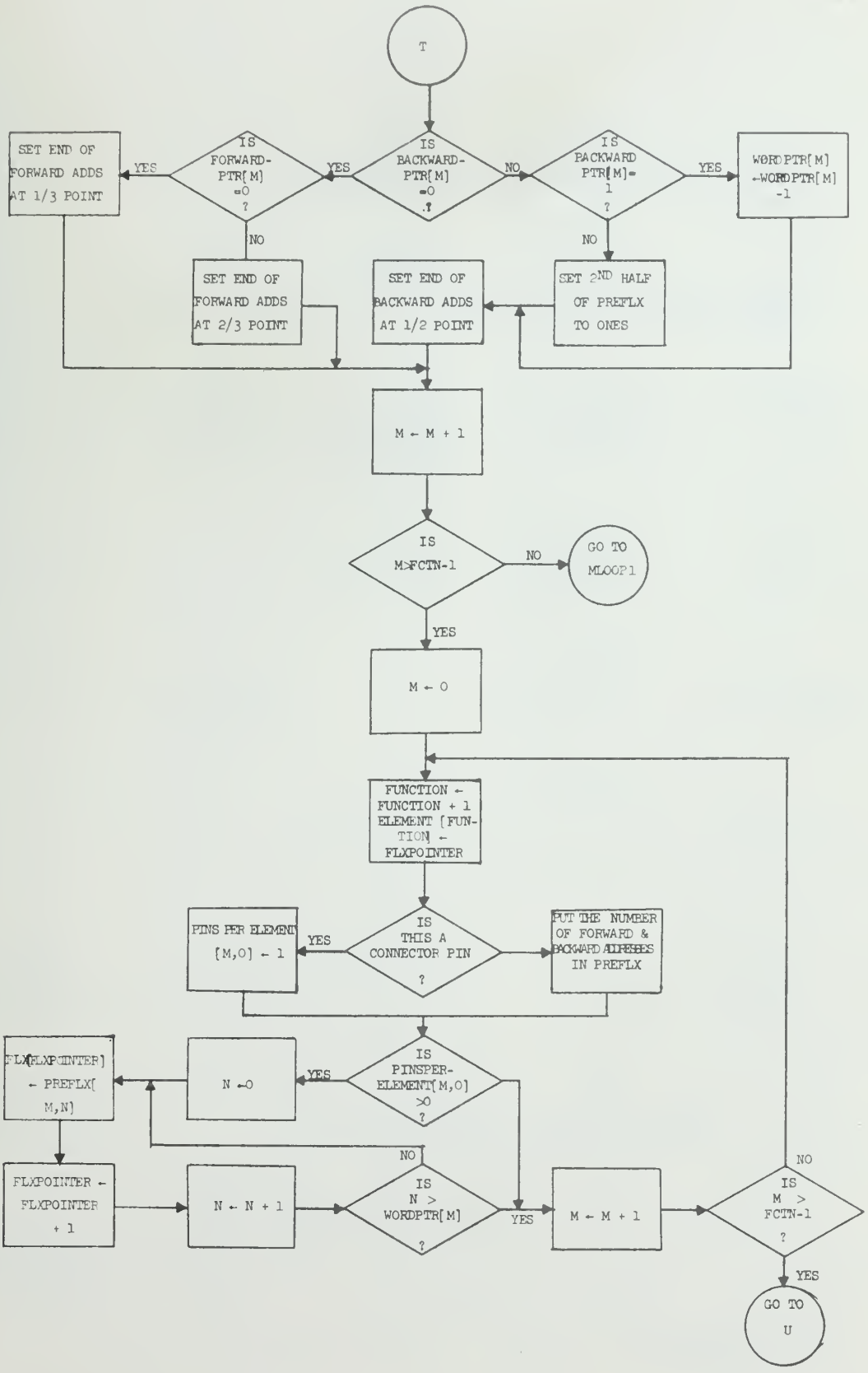
SORTINTOELEMENTS

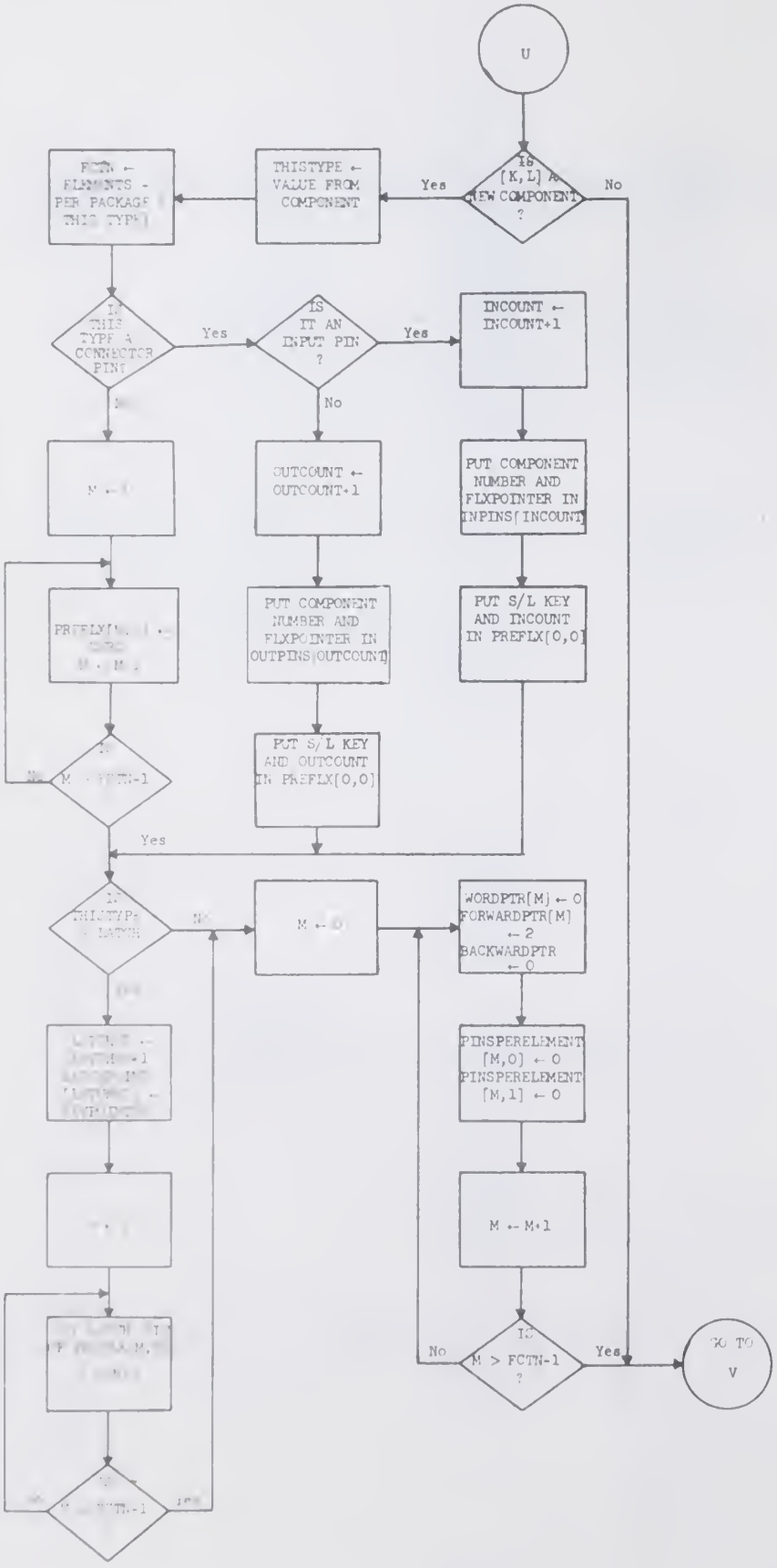

Q
M ← 1
ELEMENT-COUNT ← ELEMENT-COUNT+1
PUT ELEMENT-COUNT INTO Mth COMPONENT
M ← M + 1
IS M > THE NUMBER OF ELEMENTS IN THIS PKG. ?
NO
YES
IS THIS TYPE A CONNECTOR PIN ?
YES
PUT PIN NUMBER IN COMPONENT
COMPID ← BLANK
PUT COMPONENT-COUNT AND PINPOSITION INTO GREX[K,L]
ZERO PINCNT ARRAY
GO TO R

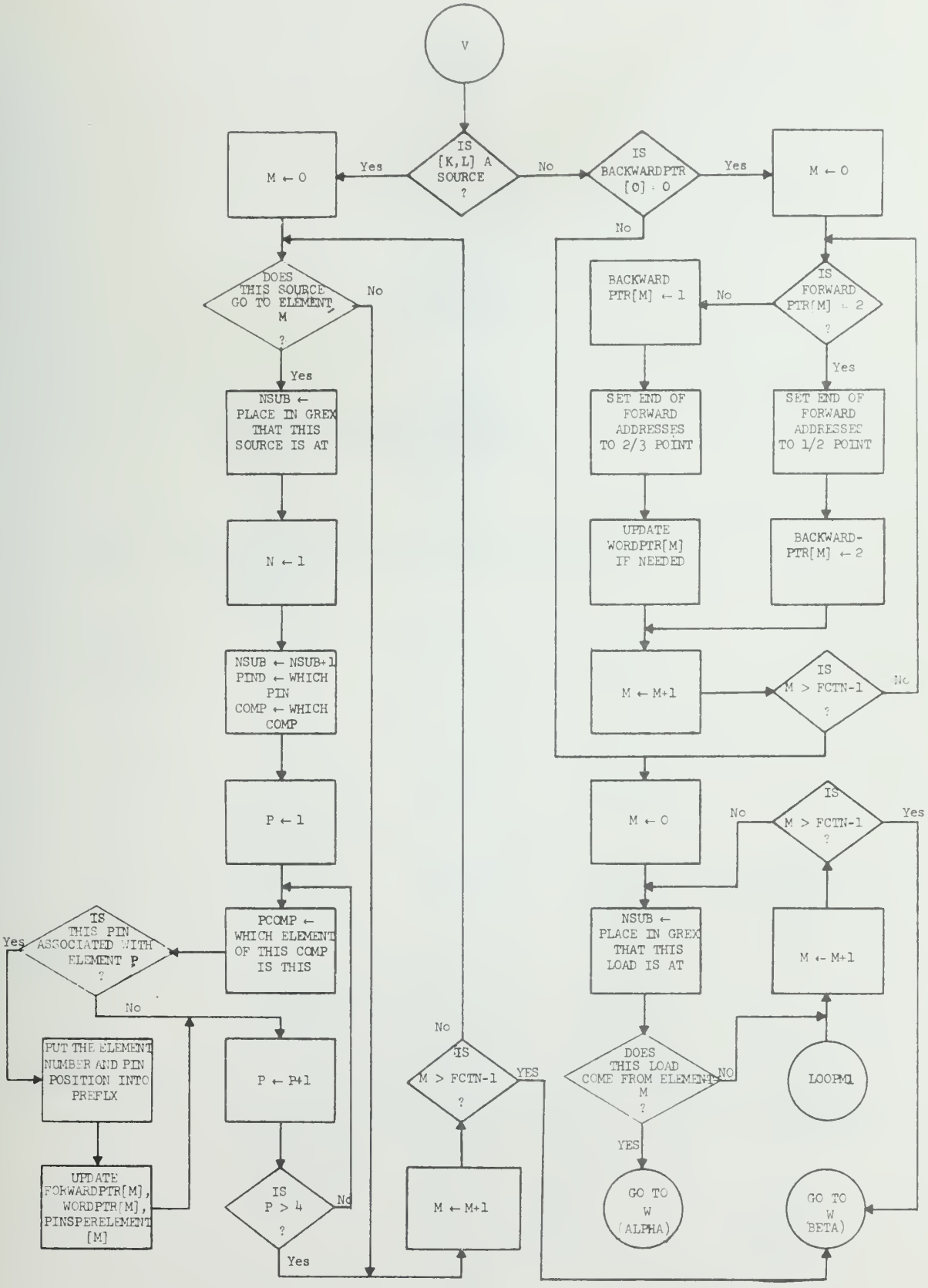

R
IS GREX[K,L] ≠ ONES ?
IS THIS A LOAD ?
Yes
IS THIS IN ELEMENT 0 ?
PUT PINCNT[0] INTO GREX[K,L]
PINCNT[0] ← PINCNT[0] + 1
No
IS THIS IN ELEMENT 1 ?
PUT PINCNT[1] INTO GREX[K,L]
PINCNT[1] ← PINCNT[1] + 1
No
IS THIS IN ELEMENT 2 ?
PUT PINCNT[2] INTO GREX[K,L]
PINCNT[2] ← PINCNT[2] + 1
No
IS THIS IN ELEMENT 3 ?
PUT PINCNT[3] INTO GREX[K,L]
PINCNT[3] ← PINCNT[3] + 1
No
J ← J+1
IS J = COLUMN[I] ?
No
GO TO JLOOP1
Yes
I ← I+1
IS I = NUMBER OF ROWS ?
No
GO TO ILOOP1
Yes
GO TO S

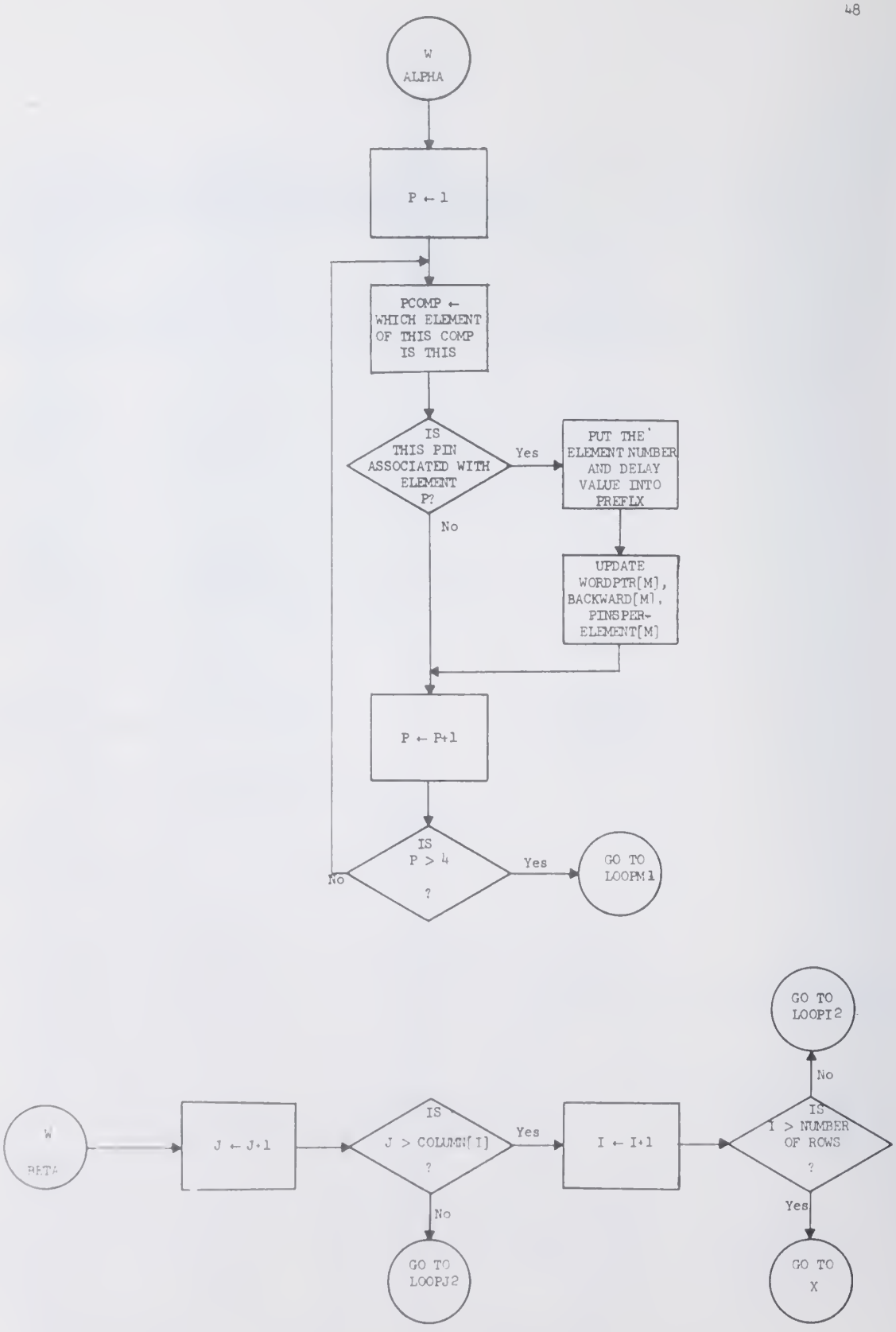


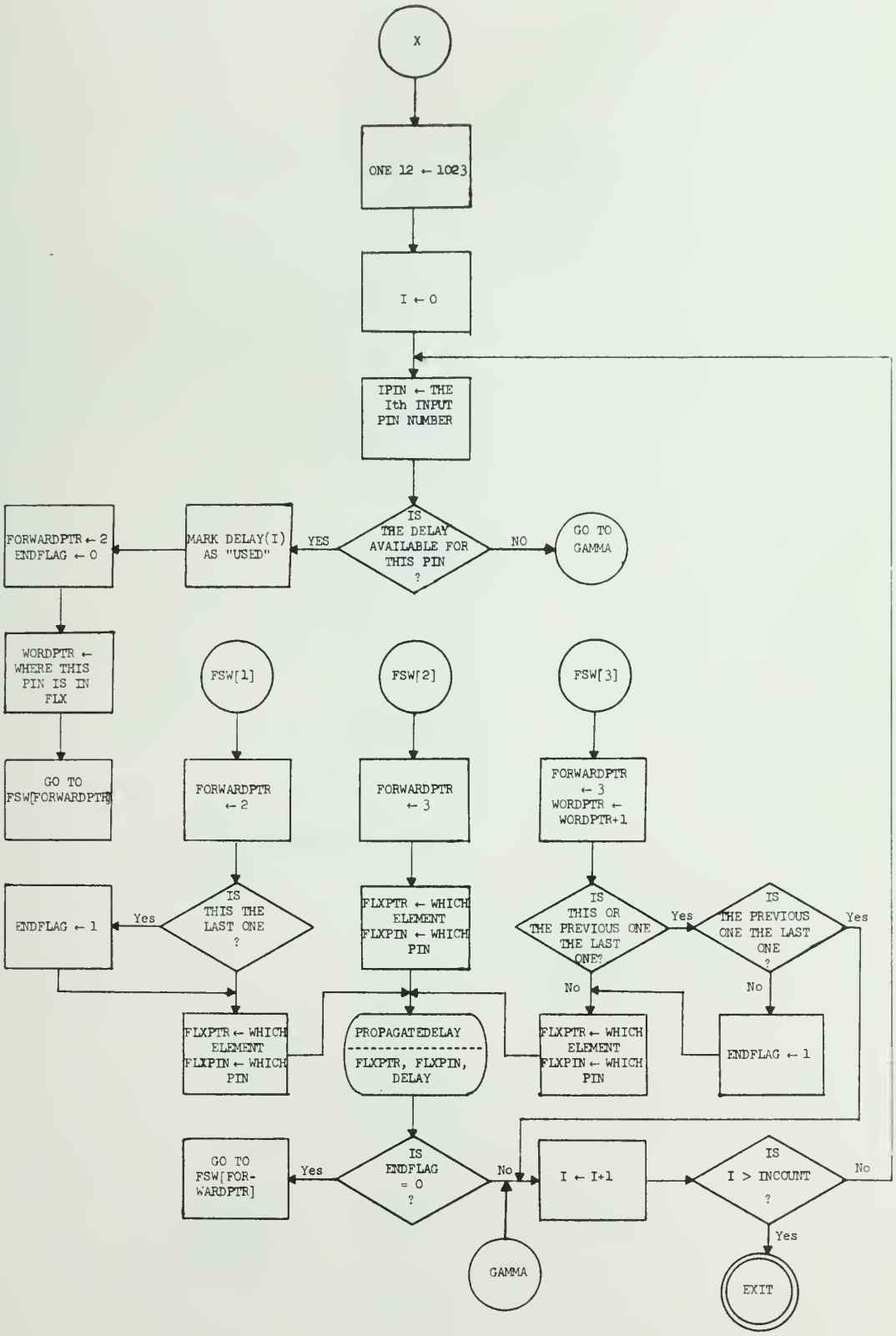
GENERATEFLX


T
IS BACKWARD-PTR[M] =0 ?
YES
IS FORWARD-PTR[M] =0 ?
YES
SET END OF FORWARD ADDS AT 1/3 POINT
NO
SET END OF FORWARD ADDS AT 2/3 POINT
NO
IS PACKWARD PTR[M]= 1 ?
YES
WORDPTR[M] ←WORDPTR[M] -1
NO
SET 2ND HALF OF PREFLX TO ONES
SET END OF BACKWARD ADDS AT 1/2 POINT
M ← M + 1
IS M>FCTN-1
NO
GO TO MLOOP1
YES
M ← 0
FUNCTION ← FUNCTION + 1 ELEMENT [FUNTION] ← FLXPOINTER
PINS PER ELEMENT [M,0] ← 1
YES
IS THIS A CONNECTOR PIN ?
PUT THE NUMBER OF FORWARD & BACKWARD ADDRESSES IN PREFLX
IS PINSPER-ELEMENT[M,0] >0 ?
YES
N ←0
FLX[FLXPOINTER] ← PREFLX[M,N]
FLXPOINTER ← FLXPOINTER + 1
N ← N + 1
NO
IS N > WORDPTR[M] ?
YES
M ← M + 1
IS M > FCTN-1 ?
NO
YES
GO TO U

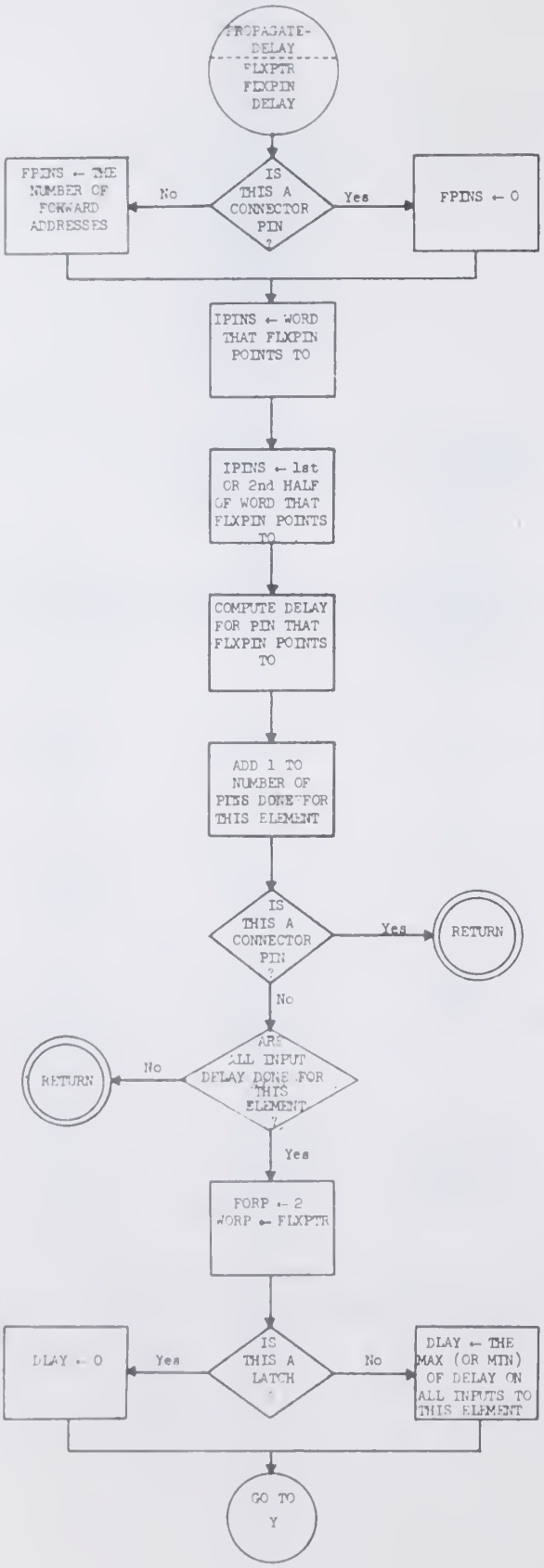

U
IS [K,L] A NEW COMPONENT ?
Yes
No
THISTYPE ← VALUE FROM COMPONENT
FCTN ← ELEMENTS PER PACKAGE [THIS TYPE]
IS THIS TYPE A CONNECTOR PIN?
Yes
IS IT AN INPUT PIN ?
Yes
No
INCOUNT ← INCOUNT+1
M ← 0
OUTCOUNT ← OUTCOUNT+1
PUT COMPONENT NUMBER AND FLXPOINTER IN INPINS[INCOUNT]
PREFLX[M,] ← 0
M ← M+1
PUT COMPONENT NUMBER AND FLXPOINTER IN OUTPINS[OUTCOUNT]
PUT S/L KEY AND INCOUNT IN PREFLX[0,0]
IS M = FCTN-1 ?
PUT S/L KEY AND OUTCOUNT IN PREFLX[0,0]
Yes
IS THISTYPE = 0 ?
M ← 0
WORDPTR[M] ← 0
FORWARDPTR[M] ← 2
BACKWARDPTR ← 0
PINSPERELEMENT [M,0] ← 0
PINSPERELEMENT [M,1] ← 0
M ← M+1
IS M > FCTN-1 ?
No
Yes
GO TO V

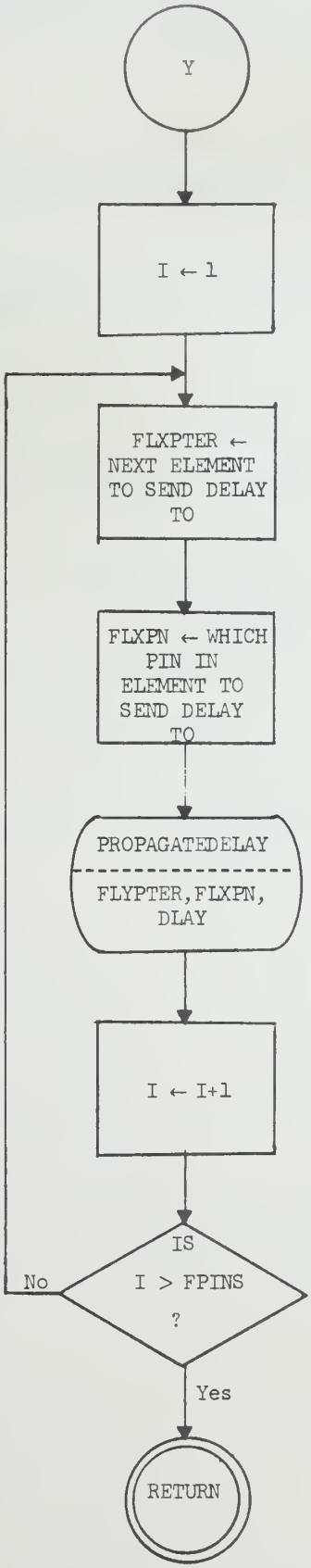

V
IS [K,L] A SOURCE ?
Yes
No
M ← 0
IS BACKWARDPTR [0] = 0
Yes
No
M ← 0
DOES THIS SOURCE GO TO ELEMENT M ?
No
Yes
NSUB ← PLACE IN GREX THAT THIS SOURCE IS AT
N ← 1
NSUB ← NSUB+1
PIND ← WHICH PIN
COMP ← WHICH COMP
P ← 1
PCOMP ← WHICH ELEMENT OF THIS COMP IS THIS
IS THIS PIN ASSOCIATED WITH ELEMENT P ?
Yes
No
PUT THE ELEMENT NUMBER AND PIN POSITION INTO PREFLX
UPDATE FORWARDPTR[M], WORDPTR[M], PINSPERELEMENT [M]
P ← P+1
IS P > 4 ?
No
Yes
IS M > FCTN-1 ?
No
YES
M ← M+1
BACKWARD PTR[M] ← 1
No
IS FORWARD PTR[M] = 2 ?
Yes
SET END OF FORWARD ADDRESSES TO 2/3 POINT
SET END OF FORWARD ADDRESSES TO 1/2 POINT
UPDATE WORDPTR[M] IF NEEDED
BACKWARD-PTR[M] ← 2
M ← M+1
IS M > FCTN-1 ?
No
M ← 0
IS M > FCTN-1 ?
No
Yes
NSUB ← PLACE IN GREX THAT THIS LOAD IS AT
M ← M+1
DOES THIS LOAD COME FROM ELEMENT M ?
NO
YES
LOOPM1
GO TO W (ALPHA)
GO TO W BETA)


W
ALPHA
P ← 1
PCOMP ← WHICH ELEMENT OF THIS COMP IS THIS
IS THIS PIN ASSOCIATED WITH ELEMENT P?
Yes
No
PUT THE ELEMENT NUMBER AND DELAY VALUE INTO PREFLX
UPDATE WORDPTR[M], BACKWARD[M], PINSPER-ELEMENT[M]
P ← P+1
IS P > 4 ?
Yes
No
GO TO LOOPM1
W
BETA
J ← J+1
IS J > COLUMN[I] ?
Yes
No
GO TO LOOPJ2
I ← I+1
IS I > NUMBER OF ROWS ?
Yes
No
GO TO LOOPI2
GO TO X


X
ONE 12 ← 1023
I ← 0
IPIN ← THE Ith INPUT PIN NUMBER
IS THE DELAY AVAILABLE FOR THIS PIN ?
YES
NO
GO TO GAMMA
MARK DELAY(I) AS "USED"
FORWARDPTR ← 2
ENDFLAG ← 0
WORDPTR ← WHERE THIS PIN IS IN FLX
GO TO FSW[FORWARDPTR]
FSW[1]
FSW[2]
FSW[3]
FORWARDPTR ← 2
FORWARDPTR ← 3
FORWARDPTR ← 3
WORDPTR ← WORDPTR+1
IS THIS THE LAST ONE ?
Yes
ENDFLAG ← 1
FLXPTR ← WHICH ELEMENT
FLXPIN ← WHICH PIN
IS THIS OR THE PREVIOUS ONE THE LAST ONE?
IS THE PREVIOUS ONE THE LAST ONE ?
No
FLXPTR ← WHICH ELEMENT FLXPIN ← WHICH PIN
PROPAGATEDELAY
FLXPTR, FLXPIN, DELAY
ENDFLAG ← 1
GO TO FSW[FORWARDPTR]
IS ENDFLAG = 0 ?
I ← I+1
IS I > INCOUNT ?
GAMMA
EXIT

SCANFLX


PROPAGATE-DELAY
FLXPTR
FLXPIN
DELAY
IS THIS A CONNECTOR PIN ?
No
Yes
FPINS ← THE NUMBER OF FORWARD ADDRESSES
FPINS ← 0
IPINS ← WORD THAT FLXPIN POINTS TO
IPINS ← 1st OR 2nd HALF OF WORD THAT FLXPIN POINTS TO
COMPUTE DELAY FOR PIN THAT FLXPIN POINTS TO
ADD 1 TO NUMBER OF PINS DONE FOR THIS ELEMENT
IS THIS A CONNECTOR PIN ?
Yes
RETURN
No
ARE ALL INPUT DELAY DONE FOR THIS ELEMENT ?
No
RETURN
Yes
FORP ← 2
WORP ← FLXPTR
IS THIS A LATCH ?
Yes
DLAY ← 0
No
DLAY ← THE MAX (OR MIN) OF DELAY ON ALL INPUTS TO THIS ELEMENT
GO TO Y


Y
I ← 1
FLXPTER ← NEXT ELEMENT TO SEND DELAY TO
FLXPN ← WHICH PIN IN ELEMENT TO SEND DELAY TO
PROPAGATEDELAY
FLYPTER,FLXPN, DLAY
I ← I+1
IS I > FPINS ?
No
Yes
RETURN

LIST OF REFERENCES

[1] Crowley, T. H., "Computers as an Aid to the Design and Manufacture of Systems," 1963 IEEE International Conv. Rec., vol. 11, pt. 4, pp. 4-51.

[2] Gill, S., "The Use of Computers in Designing Computers," Industrial Research, vol. 15, pp. 159-163, April, 1962.

[3] Gordon, W. L., "Data Processing Techniques in Design Automation," 1960 Proc. EJCC, pp. 205-209.

[4] Gray, S. R. and Kisch, R. N., "A Progress Report on Computer Applications in Computer Design," 1956 Proc. WJCC.

[5] Hanning, W. A. and Mayes, T.L., "Impact of Automation on Digital Computer Design," 1960 Proc. EJCC, pp. 211-232.

[6] Kurtzberg, J., "Computer Mechanization of Design Procedures," Proc. of the Detroit Conf. of the AIIE, October, 1964.

[7] Leichner, G. H., "Designing Computer Circuits with a Computer," J. ACM, vol. 4, pp. 143-147, April, 1957.

[8] Warshawsky, E. H., "Design Automation," Datamation, pp. 25-28, June, 1964.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

AN ALGORITHM FOR DELAY CHECKING COMPUTER DESIGNS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Research Report

5. AUTHOR(S) *(First name, middle initial, last name)*

Jay Merrill Tummelson

6. REPORT DATE
January 13, 1969

7a. TOTAL NO. OF PAGES
57

7b. NO. OF REFS
8

8a. CONTRACT OR GRANT NO.
46-26-15-305

b. PROJECT NO.
USAF 30(602)4144

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

DCS Report No. 301

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies of this report from DCS.

11. SUPPLEMENTARY NOTES

NONE

12. SPONSORING MILITARY ACTIVITY
Rome Air Development Center
Griffiss Air Force Base
Rome, New York 13440

13. ABSTRACT

The problem of delay checking computer designs is discussed along with its relation to the design automation problem as a whole. The ILLIAC IV design automation package is described as an example of systems in general. The remainder of the paper describes in detail the delay check algorithm and computer program developed by the author.

Detailed description of the data formats, internal structuring of data and flowcharts of the program are included for those interested in the application of the algorithm.

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
ILLIAC IV Design Automation System						
Delay Check Algorithm						